UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

# BIBLIOTECA PARA DESENVOLVIMENTO DE JOGOS MULTIJOGADORES DE REALIDADE AUMENTADA PARA ANDROID

DIEGO ANDRÉ LIRA

BLUMENAU
2015

2015/1-07

**DIEGO ANDRÉ LIRA**

# BIBLIOTECA PARA DESENVOLVIMENTO DE JOGOS MULTIJOGADORES DE REALIDADE AUMENTADA PARA ANDROID

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

Prof. Dalton Solano dos Reis, M. Sc. - Orientador

**BLUMENAU**
**2015**

**2015/1-07**

# BIBLIOTECA PARA DESENVOLVIMENTO DE JOGOS MULTIJOGADORES DE REALIDADE AUMENTADA PARA ANDROID

Por

**DIEGO ANDRÉ LIRA**

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

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

Membro: ______________________________________________________
Prof. Mauro Marcelo Mattos, Doutor – FURB

Membro: ______________________________________________________
Prof. Maurício Capobianco Lopes, Doutor – FURB

Blumenau, 07 de julho de 2015

Dedico este trabalho à minha esposa Léia pela incansável dedicação, inexplicável paciência e, por algum mistério do universo, inesgotável paixão.

# AGRADECIMENTOS

A Deus pelas oportunidades.

À minha família pelas lições valorosas e apoio.

À minha esposa por ser a força motriz por trás de cada investida.

Ao meu orientador pela dedicação, tempo e interesse investido neste trabalho.

Sucesso consiste em ir de fracasso a fracasso sem perder o entusiasmo.

Winston Churchill

# RESUMO

Este trabalho descreve o desenvolvimento de uma biblioteca para a criação de jogos multijogadores de Realidade Aumentada (RA) na plataforma Android, denominada FurbRA-library. A biblioteca desenvolvida é baseada na biblioteca Vuforia, para a construção e iteração com RA, na biblioteca OpenGL ES, para a construção e desenho dos objetos virtuais e nas bibliotecas da plataforma Android para a comunicação, multimídia e interação homem máquina, através da tela sensível a toques. A biblioteca permite que sejam desenvolvidos jogos que envolvem o conceito de RA com marcadores gráficos, sem exigir do desenvolvedor um conhecimento profundo de material de base, tal como conhecimento da biblioteca Vuforia e com o sistema operacional. A biblioteca resultante atingiu os principais requisitos propostos, sendo, ao final deste, obtido um jogo, criado utilizando a FurbRA-library como base.



Palavras-chave: Android. Realidade aumentada. Mobilidade. Comunicação.

# ABSTRACT

This work describes the development of a library for building Augmented Reality (AR) multiplayer games on the Android platform, called FurbRA-library. The developed library is based on Vuforia library for the construction and iteration with RA, OpenGL ES library for the construction and design of virtual objects and with Android platform libraries for communication, multimedia and human machine interaction through touch screen. The library allows games that involve the concept of graphic markers AR be develop, without requiring the developer a thorough knowledge of basic material, such as knowledge of Vuforia library and the operating system. The resulting library reached the main proposed requirements and, at the end of this, was possiblte to obtain a game created using the FurbRA-library as a basis.

Key-words: Android. Augmented reality. Mobility. Communication.

# LISTA DE FIGURAS

Figura 1 - Publicidade do jogo Pong ....... 21
Figura 2 - Marcador do tipo frame ....... 27
Figura 3 - Extended Tracking ....... 28
Figura 4 - Aplicativo de Vasselai (2010) em funcionamento ....... 29
Figura 6 - Visão de camada da FurbRA-library ....... 32
Figura 7 - Diagrama de casos de uso ....... 32
Figura 8 - Diagrama de pacotes da FurbRA-library ....... 36
Figura 9 - Diagrama de classe do pacote `ra` ....... 36
Figura 10 - Diagrama de classe do pacote `render` ....... 37
Figura 11 - Diagrama de classe do pacote `config` ....... 38
Figura 12 - Diagrama de classes do pacote `media` ....... 39
Figura 13 - Diagrama de classes do pacote `communication` ....... 40
Figura 14 - Digrama de classes do pacote `activity` ....... 42
Figura 15 - Digrama de classes do pacote `utils` ....... 43
Figura 16 - Diagrama de sequência da obtenção de instâncias do aplicativo na rede local ....... 44
Figura 17 - Diagrama de sequência do processo de criação da RA ....... 45
Figura 18 - Tela inicial da aplicação Animais ....... 77
Figura 19 - Tela de menu da aplicação Animais ....... 78
Figura 20 - Tela sobre da aplicação Animais ....... 79
Figura 21 - Tela de carregamento do jogo *multiplayer* da aplicação Animais ....... 80
Figura 22 - Aplicação Animais ....... 81
Figura 23 - Diagrama de pacotes da aplicação Animais ....... 81
Figura 24 - Tela de definição da activity inicial do projeto ....... 83
Figura 25 - Importação da biblioteca FurbRA-library ....... 84
Figura 26 - Inclusão da biblioteca como dependência ....... 84
Figura 27 - Inclusão das dependências da biblioteca Vuforia ....... 85
Figura 28 – Arquivo de configurações furbra_config.xml ....... 85
Figura 29 - Gráfico do impacto da quantidade de objetos no desempenho gráfico da aplicação ....... 92
Figura 30 - Gráfico do impacto da quantidade de mensagens enviadas/recebidas ....... 94
Figura 31 - Gabarito paras as peças do jogo Animais ....... 101

# LISTA DE QUADROS

Quadro 1 - Caso de uso UC01 .......... 33
Quadro 2 - Caso de uso UC02 .......... 33
Quadro 3 - Caso de uso UC03 .......... 33
Quadro 4 - Caso de uso UC04 .......... 34
Quadro 5 - Caso de uso UC05 .......... 34
Quadro 6 - Caso de uso UC06 .......... 34
Quadro 7 - Caso de uso UC07 .......... 35
Quadro 8 - Caso de uso UC08 .......... 35
Quadro 9 - Caso de uso UC09 .......... 35
Quadro 10- Implementação do método construtor da classe `FurbRaConfig` .......... 46
Quadro 11 - Implementação do método `initAR` .......... 47
Quadro 12 - Implementação do método `doInBackGround` da *inner class* `InitVuforiaTask` .......... 48
Quadro 13 - Implementação do método `onPostExecute` da *inner class* `InitVuforiaTask` .......... 49
Quadro 14 - Implementação do método `doInBackground` da *inner class* `LoadTrackerTask` .......... 50
Quadro 15 - Implementação do método `onPostExecute` da *inner class* `LoadTrackerTask` .......... 51
Quadro 16 - Implementação do método startAR da classe VuforiaUpdateCallback .......... 52
Quadro 17 - Implementação do método `pauseAR` da classe `VuforiaUpdateCallback` .......... 54
Quadro 18 - Implementação do método `stopAR` da classe `VuforiaUpdateCallback`. 54
Quadro 19 - Implementação do método `resumeAR` da classe `VuforiaUpdateCallback` .......... 55
Quadro 20 - Implementação do método `onCreate` da classe `FurbRaBaseActivity` ... 55
Quadro 21 - Implementação do método `onResume` da classe `FurbRaBaseActivity` ... 56
Quadro 22 - Implementação do método `onInitARDone` da classe `FurbRaBaseActivity` .......... 56
Quadro 23 - Implementação do método `onPause` da classe `FurbRaBaseActivity` ...... 57

Quadro 24 - Implementação do método `onDestroy` da classe `FurbRaBaseActivity`.57
Quadro 25 - Implementação do método `showToast` da classe `FurbRaBaseActivity`.57
Quadro 26 - Implementação do método `getDisplayMetrics` da classe `FurbRaBaseActivity` ....57
Quadro 27 - Métodos abstratos que definem o gerenciamento de marcadores nas subclasses de `FurbRaBaseActivity` ....58
Quadro 28 - Métodos abstratos utilizados nas notificações de RA nas subclasses de `FurbRaBaseActivity` ....58
Quadro 29 - Métodos abstratos utilizados no processo de renderização nas subclasses de `FurbRaBaseActivity` ....59
Quadro 30 - Implementação do método `addMarker` da classe `FurbRaFrameActivity` ....59
Quadro 31 - Implementação do método `removeMarker` da classe `FurbRaFrameActivity` ....60
Quadro 32 – Implementação do método `doInitTrackers` da classe `FurbRaFrameActivity` ....60
Quadro 33 - Implementação do método `doLoadTrackersData` da classe `FurbRaFrameActivity` ....61
Quadro 34 - Implementação do método `doStartTrackers` da classe `FurbRaFrameActivity` ....61
Quadro 35 - Implementação do método `doStopTrackers` da classe `FurbRaFrameActivity` ....61
Quadro 36 - Implementação do método `doUnloadTrackersData` da classe `FurbRaFrameActivity` ....62
Quadro 37 - Implementação do método `doDeinitTrackers` da classe `FurbRaFrameActivity` ....62
Quadro 38 - Implementação do método `doInitTrackers` da classe `FurbRaTextRecognitionActivity` ....63
Quadro 39 - Implementação do método `doStartTrackers` da classe `FurbRaTextRecognitionActivity` ....63
Quadro 40 - Implementação do método `doStopTrackers` da classe `FurbRaTextRecognitionActivity` ....63

Quadro 41 - Implementação do método doUnloadTrackersData da classe FurbRaTextRecognitionActivity ........ 64
Quadro 42 - Implementação do método doDeinitTrackers da classe FurbRaTextRecognitionActivity ........ 64
Quadro 43 - Implementação do método doLoadTrackersData da classe FurbRaTextRecognitionActivity ........ 65
Quadro 44 - Implementação do método construtor da classe FurbRaGLView ........ 66
Quadro 45 - Implementação do método onSurfaceCreated da classe FurbRaRenderer ........ 66
Quadro 46 - Implementação do método onSurfaceChanged da classe FurbRaRenderer ........ 66
Quadro 47 - Implementação do método onDrawFrame da classe FurbRaRenderer ........ 67
Quadro 48 - Implementação do método renderFrame da classe FurbRaRenderer ........ 68
Quadro 49 - *Inner class* LoadSoundTask ........ 69
Quadro 50 - Implementação do método startService da classe LocalNetworkCommunication ........ 70
Quadro 51 - Método initializeRegistrationListener da classe LocalNetworkCommunication ........ 71
Quadro 52 - Implementação do método startDiscovery da classe LocalNetworkCommunication ........ 72
Quadro 53 – Implementação do método initializeDiscoveryListener da classe LocalNetworkCommunication ........ 73
Quadro 54 - *Inner class* ServerThread ........ 74
Quadro 55 – *Inner class* IncomeCommunicationThread ........ 75
Quadro 56 - I*nner class* OutcomeCommunicationThread ........ 76
Quadro 57 - Permissões necessárias a utilização dos recursos da FurbRA-library ........ 86
Quadro 58 - Implementação do método onCreate da classe MultiGameActivity ........ 87
Quadro 59 - Implementação do método loadSounds da classe MultiGameActivity ........ 88
Quadro 60 - Implementação do método loadTextures da classe MultiGameActivity ........ 88
Quadro 61 - Implementação do método registerCommListeners da classe MultiGameActivity ........ 89

Quadro 62 - Implementação do método `trackerFound` da classe `MultiGameActivity` ....................90
Quadro 63 - Implementação do método `wordFinded` da classe `MultiGameActivity`.91
Quadro 64 - Comparação com os trabalhos correlatos....................95
Quadro 65- Formulário de avaliação da experiência de usuário da aplicação Animais.........102

# LISTA DE TABELAS

Tabela 1 - Medição da taxa de FPS na renderização de objetos................................................ 92
Tabela 2 - Medição de tempo entre o envio e recebimento de mensagens .............................. 93

# LISTA DE ABREVIATURAS E SIGLAS

API – *Application Programming Interface*

IDE - *Integrated Development Environment*

I/O – *Input/Output*

LIBRAS – LInguagem BRAsileira de Sinais

POJO – *Plain Old Java Object*

RA – Realidade Aumentada

## SUMÁRIO


**1 INTRODUÇÃO ... 19**
1.1 OBJETIVOS ... 19
1.2 ESTRUTURA ... 20
**2 FUNDAMENTAÇÃO TEÓRICA ... 21**
2.1 JOGOS MULTIJOGADORES ... 21
2.2 REALIDADE AUMENTADA ... 22
2.3 ANDROID ... 23
2.3.1 Sensores ... 23
2.3.2 Conectividade ... 24
2.3.3 Multimídia ... 24
2.3.4 Activities ... 25
2.3.5 OpenGL ES ... 26
2.4 VUFORIA ... 26
2.5 TRABALHOS CORRELATOS ... 28
2.5.1 Um Estudo Sobre Realidade Aumentada para a Plataforma Android ... 28
2.5.2 ARToolKit ... 29
2.5.3 Unreal Engine ... 30
**3 DESENVOLVIMENTO DA BIBLIOTECA ... 31**
3.1 REQUISITOS PRINCIPAIS DA BIBLIOTECA A SER DESENVOLVIDA ... 31
3.2 ESPECIFICAÇÃO ... 31
3.2.1 Visão da biblioteca ... 31
3.2.2 Casos de Uso ... 32
3.2.3 Diagrama de Classes ... 35
3.2.3.1 Pacote `ra` ... 36
3.2.3.2 Pacote `render` ... 37
3.2.3.3 Pacote `config` ... 37
3.2.3.4 Pacote `media` ... 38
3.2.3.5 Pacote `communication` ... 40
3.2.3.6 Pacote `activity` ... 42
3.2.3.7 Pacote `utils` ... 42
3.2.4 Diagrama de sequência ... 44

3.3 IMPLEMENTAÇÃO ........ 45
3.3.1 Técnicas e ferramentas utilizadas ........ 45
3.3.2 Configurações da biblioteca ........ 46
3.3.3 A integração com a biblioteca de RA ........ 46
3.3.4 `Activity` base ........ 55
3.3.5 *Activity* para detecção de marcadores *Frame* ........ 59
3.3.6 `Activity` para detecção de textos ........ 62
3.3.7 Renderização ........ 65
3.3.8 Mídia ........ 68
3.3.9 Comunicação ........ 69
3.3.10 Operacionalidade da implementação ........ 76
3.3.10.1 Visão geral da aplicação ........ 76
3.3.10.1.1 Tela inicial ........ 77
3.3.10.1.2 Menu ........ 77
3.3.10.1.3 `Sobre` ........ 78
3.3.10.1.4 `Sem amiguinhos` ........ 79
3.3.10.1.5 `Com amiguinhos` ........ 79
3.3.10.2 Estrutura de classes ........ 81
3.3.10.2.1 Pacote `model` ........ 82
3.3.10.2.2 Pacote `mesh` ........ 82
3.3.10.2.3 Pacote `view` ........ 82
3.3.10.2.4 Pacote `comm` ........ 82
3.3.10.2.5 Pacote `activity` ........ 83
3.3.10.3 Início do projeto ........ 83
3.3.10.4 Utilizando a FurbRA-library ........ 85
3.4 RESULTADOS E DISCUSSÕES ........ 91
3.4.1 Dificuldades encontradas ........ 94
3.4.2 Comparativo entre a biblioteca desenvolvida e seus correlatos ........ 94
**4 CONCLUSÕES ........ 96**
4.1 EXTENSÕES ........ 97
**REFERÊNCIAS ........ 98**
**APÊNDICE A – Gabarito paras as peças do jogo Animais ........ 101**

**APÊNDICE B – Formulário de avaliação da experiência de usuário da aplicação Animais 102**

# 1 INTRODUÇÃO

É previsto que a indústria de aplicativos de Realidade Aumentada irá valorizar-se até o patamar de cerca de U$5.151,74 milhões até 2016, tendo assim uma taxa composta de crescimento de 95,3% ao ano (MARKETSANDMARKETS, 2014). A Realidade Aumentada está sendo introduzida nas mais diversas áreas como jogos, indústria automotiva, medicina, propaganda, defesa, *e-learning* e navegação (MARKETSANDMARKETS, 2014).

Com cada vez mais empresas investindo em dispositivos que não só permitem, mas possuem foco em Realidade Aumentada, é possível prever um forte aquecimento no mercado de aplicativos que se utilizam desta tecnologia. O projeto Glass da Google, pode ser visto como um dos pontos base dessa previsão. Este projeto demonstra como a tecnologia de Realidade Aumentada possui um apelo não só para a indústria, mas também popular. Antes mesmo de ser lançado, o Google Glass foi citado como uma das melhores invenções de 2012 pela revista Times Magazine (YEUNG, 2012). Utilizando-se da plataforma Android, em acréscimo do Glass Development Kit (GDK), será possível utilizar recursos e *frameworks* já conhecidos da plataforma Android para criar aplicações que integram a realidade com informações virtuais.

Enquanto dispositivos são planejados para o futuro, aplicativos mais simples que utilizam os conceitos de Realidade Aumentada, já estão disponíveis atualmente. O jogo AR Defender 2 (INT13, 2012), lançado em 2012 para as plataformas mobile iOS e Android, utiliza a Realidade Aumentada de forma a permitir imersão por parte do usuário no jogo. Imersão esta que é fortemente acrescentada pelo fato de possibilitar até quatro jogadores jogarem simultaneamente uma partida cooperativa em uma rede local sem fio. Este recurso possibilita que não só o jogador sinta o jogo como parte da realidade, mas as pessoas que o cercam, e participam do jogo também estão presentes e interagem com esta realidade tornando-a mais crível.

Com o exposto acima, este trabalho tem como objetivo desenvolver uma biblioteca que auxilie o desenvolvimento de jogos de Realidade Aumentada para Android.

## 1.1 OBJETIVOS

O objetivo deste trabalho é desenvolver uma biblioteca de RA para a plataforma Android.

Os objetivos específicos do trabalho são:

a) disponibilizar um conjunto de funções e recursos de forma a facilitar a criação da Realidade Aumentada baseada em marcadores;

b) disponibilizar um conjunto de funções e recursos de forma a facilitar o gerenciamento do áudio;
c) disponibilizar um conjunto de funções e recursos de forma a facilitar a utilização da biblioteca OpenGL ES necessária no desenvolvimento gráfico dos jogos de Realidade Aumentada;
d) disponibilizar um conjunto de funções e recursos de forma a facilitar a comunicação local entre os aparelhos através da especificação de rede WiFi.

## 1.2 ESTRUTURA

A estrutura deste trabalho está apresentada em quatro capítulos.

O segundo capítulo contém a fundamentação teórica necessária para o entendimento deste trabalho.

O terceiro capítulo apresenta como foi desenvolvida a biblioteca na plataforma Android, os casos de uso, os diagramas de classe e toda a especificação que define a biblioteca. Ainda no terceiro capítulo são apresentadas as partes principais da implementação e também os resultados e discussões que aconteceram durante toda a etapa de desenvolvimento do trabalho.

Por fim, o quarto capítulo refere-se às conclusões do presente trabalho e sugestões para trabalhos futuros.

# 2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 são apresentados conceitos de jogos multijogadores. A seção 2.2 descreve o conceito de Realidade Aumentada e suas características. Na seção 2.3 é apresentada a plataforma Android e sua organização. A seção 2.4 descreve a *Application Programming Interface* (API) Vuforia (QUALCOMM, 2014a). Por fim, a seção 2.5 descreve os trabalhos correlatos.

## 2.1 JOGOS MULTIJOGADORES

A história dos jogos multijogadores, ou jogos *multiplayers* como são mais conhecidos, tem sua trajetória iniciada na década de 70 com um dos primeiros, e possivelmente mais famosos, jogos multiplayer da história: Pong. Desenvolvido por Al Alcorn para a recém-fundada Syzygy/Atari em junho de 1972, Pong foi especificado por Nolan Bushnell como sendo o jogo mais simples idealizável, sendo este um jogo de tênis (LOWOOD, 2009). Pong, em sua primeira versão para *arcade*, possuía dois controles de maneira que dois jogadores controlavam, cada um, uma barra na tela que simulava uma raquete. Esta funcionalidade inclusive era utilizada como um diferencial do jogo em sua publicidade, conforme demonstrado na Figura 1.

Figura 1 - Publicidade do jogo Pong

Fonte: Lowood (2009).

Deixando para trás o conceito de poucos jogadores, os *Multi Massive Online Role-Playing Game* (MMORPG) arrebatam milhões de usuários em partidas simultâneas. World of

Warcraft, MMORPG da produtora Blizard, atingiu picos de 12 milhões de assinaturas ativas em 2010 (STATISTA, 2014). A indústria de jogos possuía expectativas de faturar US$ 66 bilhões em 2013 (REUTERS, 2013), sendo que deste total o MMORPG League of Legends da produtora Riot Games faturou sozinho US$ 624 milhões (HANDRAHAN, 2014), cerca de 1% da expectativa total do mercado.

Enquanto até mesmo nas plataformas mobile é possível jogar simultaneamente com pessoas espalhadas ao redor do mundo, ainda é possível encontrar casos de jogos que se utilizam de proximidade para o seu desenvolvimento. Estes jogos, conectados em redes pessoais sem fio ou com tecnologias de curto alcance, exigem que os jogadores estejam no mesmo ambiente de modo a compartilhar uma partida. Destes jogos podem ser citados Drive on Moscow (SHENANDOAH, 2014), Pandemic: The Board Game (F2Z, 2014) e AR Defender 2 (INT13, 2012).

## 2.2 REALIDADE AUMENTADA

Segundo Azuma (1997, p.2), a Realidade Aumentada é uma variação da Realidade Virtual. Enquanto na Realidade Virtual a pessoa é o objeto real inserido em contexto virtual, a Realidade Aumentada sobrepõe ou compõe a realidade com objetos virtuais.

Azuma (1997, p.3) especifica a motivação desta tecnologia como uma melhoria na percepção e interação com o mundo virtual auxiliando, através dos objetos virtuais, o usuário a executar ações e tarefas no mundo real. Azuma (1997, p.2) define três propriedades básicas que definem um sistema como sendo Realidade Aumentada, sendo estas descritas a seguir.

A primeira propriedade especificada por Azuma (1997, p.2) é a combinação do real com o virtual. Azuma et al. (2001, p1) define esta propriedade como a combinação precisa entre objetos virtuais e reais e define que sem esta precisa combinação, a ilusão de coexistência do virtual com o real é severamente comprometida. Esta propriedade exige algum tipo de hardware específico, mas não se limita a uma especificação tecnológica (AZUMA, 1997, p.2) ou até mesmo ao sentido visual (AZUMA et al., 2001, p.1), sendo abrangente aos demais sentidos como tato ou audição.

A segunda propriedade definida por Azuma (1997, p.2) é a interação em tempo real. Segundo Buchmann et al. (2004, p.1) a interação com a Realidade Aumentada deve ser a mais intuitiva possível. No passado esta interação ocorreu com uma variedade de métodos como a identificação de marcadores, mouse e teclado, entre outros, mas houve pouco avanço em técnicas de interação utilizando somente as mãos. Buchmann et al. (2004, p.1) descreve a ação sobre a Realidade Aumentada pelo movimento de mãos como sendo o principal método

de entrada para interfaces de Realidade Aumentada em vista que as mãos já são o principal método de interação com a realidade para uma pessoa. O problema desta técnica é quanto a identificação das mãos do usuário e do movimento realizado. Durante o desenvolvimento da aplicação FingARtips, Buchmann et al. (2004) tiveram que utilizar uma luva com marcadores nos dedos de maneira que a mão pudesse ser reconhecida pelo sistema.

A terceira propriedade definida por Azuma et al. (2001, p1) trata da sobreposição das informações virtuais que deve manter a noção 3D existente no mundo real. Esta propriedade apresenta um grande desafio em vista que câmeras convencionais fornecem representações 2D da realidade, portanto não possuem noção de profundidade. Uma das técnicas para contornar esta situação, inclusive tendo sido utilizada por Buchmann et al. (2004) em sua aplicação FingARtips, é a utilização de marcadores com tamanhos pré-definidos de forma a poder haver alguma informação na imagem obtida pela câmera para se utilizar como base para uma escala de tamanhos e orientação.

## 2.3 ANDROID

Android é um sistema operacional para dispositivos móveis baseado em Linux (KARCH, 2014). Segundo Google (2014a), a plataforma possui atualmente a maior base instalada entre os dispositivos móveis, havendo cerca de um milhão de novas ativações todos os dias. Dentre os recursos oferecidos pela plataforma pode-se citar o suporte a sensores, conectividade, multimídia, *activities* e a biblioteca OpenGL ES.

### 2.3.1 Sensores

Segundo Google (2014i), os dispositivos Android possuem sensores que podem ser divididos em três categorias:

a) sensores de posicionamento: os sensores de posicionamento fornecem informações baseadas no posicionamento físico do dispositivo. Estas informações são providas ao Android por dispositivos como sensores de orientação e magnetômetros existentes no dispositivo;
b) sensores de movimento: os sensores de movimento medem forças de aceleração e rotação exercidas sobre o aparelho. Esta categoria inclui os acelerômetros, sensores de gravidade e giroscópios;
c) sensores de condições ambientais: estes sensores medem fatores como temperatura, pressão, umidade e iluminação. Com exceção do sensor de iluminação, normalmente utilizado pelos fabricantes para ajustar o brilho da tela

do dispositivo, os sensores de condições ambientais não são incluídos nos aparelhos mais populares.

O acesso a estes sensores, através da plataforma Android, ocorre pelo *framework* Sensor, que possui uma série de classes e interfaces que auxiliam o desenvolvedor através de tarefas como a obtenção da quantidade e tipos de sensores disponíveis em um aparelho e a determinação de capacidades individuais destes como alcance máximo, requisito energético e resolução.

### 2.3.2 Conectividade

A plataforma Android disponibiliza uma série de funcionalidades que permite que um dispositivo Android conecte-se e interaja com outros dispositivos (GOOGLE, 2014d). Entre as tecnologias de conectividade existentes na plataforma Android pode-se citar o Bluetooth, o Near Field Communication (NFC) e o Network Service Discovery (NSD):

a) Bluetooth: a tecnologia Bluetooth permite a troca de informações sem fio ponto a ponto ou multiponto entre aparelhos que implementam esta tecnologia. O acesso às classes e bibliotecas responsáveis por criar uma comunicação Bluetooth ocorre através do pacote `android.bluetooth`, sendo este nativo da plataforma (GOOGLE, 2014b);
b) Near Field Communication: a NFC é caracterizada como sendo uma tecnologia de comunicação sem fio de curto alcance, tipicamente necessitando de uma distância de 4 centímetros ou menos. Este tipo de comunicação pode ocorrer entre dois dispositivos Android ou entre um dispositivo Android e uma tag NFC (GOOGLE, 2014g);
c) Network Service Discovery: o NSD por si não implementa nenhuma forma de comunicação entre dispositivos e sim permite a publicação, e identificação, de serviços *peer-to-peer* na rede local (GOOGLE,2015c).

### 2.3.3 Multimídia

O *framework* de multimídia do Android oferece suporte a uma variedade de tipos comuns de mídia, facilitando a integração de áudio, vídeo e imagens nas aplicações desenvolvidas para a plataforma (GOOGLE, 2014f). A utilização dos recursos de áudio e vídeo ocorrem através da classe `MediaPlayer`, que oferece suporte tanto para a execução de arquivos locais, presentes no dispositivo, quanto para a execução de mídias obtidas por *stream* em conexões de rede.

Na utilização das câmeras presentes nos dispositivos Android o *framework* de multimídia apresenta duas classes principais: a classe `Camera`, que é utilizada para o controle das câmeras do dispositivo, podendo tirar fotos ou gravar vídeos, e a classe `SurfaceView` utilizada para visualizar a imagem ao vivo da câmera (GOOGLE, 2014c).

### 2.3.4 Activities

Uma `activity` é um componente que providencia uma tela com o qual o usuário pode interagir. A cada `activity` é dada, pelo sistema operacional, uma janela na qual é renderizada a interface com o usuário. Normalmente, e principalmente em dispositivos menores, essa janela preenche a tela, embora possa ser menor que a tela ou até mesmo flutuar por cima de outra janela. Uma aplicação em Android normalmente consiste em múltiplas `activities` que não se relacionam diretamente (GOOGLE, 2015b).

Segundo Google (2015b) o ciclo de vida de uma `activity` é divido em três estados:

a) em execução: a `activity` está em primeiro plano na tela e tem o foco do usuário;
b) pausada: outra `activity` está em primeiro plano, mas a `activity` ainda é visível, mesmo que parcialmente;
c) parada: a `activity` está completamente obscurecida por outra `activity`. Nesse estado a `activity` pode ser encerrada pelo sistema operacional em razão da necessidade de memória.

A plataforma Android fornece uma série de *callbacks* a fim de notificar a `activity` quanto às mudanças no *status* de execução da aplicação. A lista de *callbacks* e seus momentos de execução estão listados a seguir:

a) `onCreate`: chamado quando a `activity` é criada. Sempre é seguido do método `onStart`;
b) `onRestart`: chamado quando a `activity` foi parada, logo antes de ser reiniciada novamente. Sempre é seguida do método `onStart`;
c) `onStart`: chamado logo antes da `activity` se tornar visível ao usuário. Seguido do método `onResume` se a `activity` vier a ficar em primeiro plano ou `onStop` se a mesma ficar oculta;
d) `onResume`: chamado logo antes da `activity` se tornar interativa com o usuário. Sempre seguida do método `onPause`;
e) `onPause`: chamado quando o sistema irá executar outra `activity`. Seguido do método `onResume` se a `activity` tornar a ficar visível ou `onStop` se a mesma se

tornar invisível ao usuário;

f) `onStop`: chamado quando a `activity` não pode ser mais visível pelo usuário. Seguido do método `onRestart` se a `activity` for retornada ou `onDestroy` se a `activity` for eliminada pelo sistema;
g) `onDestroy`: chamado quando a `activity` for finalizada. Essa é a última chamada que a `activity` irá receber.

### 2.3.5 OpenGL ES

Segundo Khronos (2014), o OpenGL ES é uma biblioteca para sistemas embarcados que permite a criação de gráficos 2D e 3D. Obtida através de um subconjunto da biblioteca OpenGL, esta consiste em uma interface entre o software e a aceleração gráfica. A especificação mais recente da OpenGL ES é a 3.0 embora dados da Google (2014e) demonstrem que, dentre os aparelhos Android, a versão 2.0 ainda consiste em 89.4% desses.

Segundo Google (2014h), na plataforma Android existem duas classes fundamentais que permitem criar e manipular gráficos através do OpenGL ES:

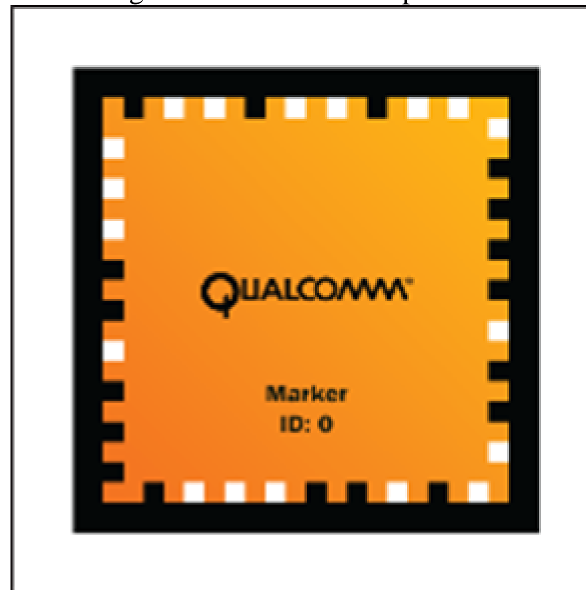
a) `GLSurfaceView`: esta classe, que herda da classe nativa `View`, possibilita a renderização e a manipulação de objetos usando chamadas OpenGL ES;
b) `GLSurfaceView.Renderer`: esta interface define os métodos necessários para a renderização dos gráficos em um objeto `GLSurfaceView`.

## 2.4 VUFORIA

Vuforia é desenvolvido para iOS, Android e Unity 3D pela fabricante de processadores Qualcomm. A biblioteca Vuforia consiste em um conjunto de funções e recursos que facilita o desenvolvimento de aplicações de Realidade Aumentada (QUALCOMM, 2014a).

A ferramenta apresenta diversas funcionalidades dentre quais pode-se destacar como uma das mais proeminentes a identificação de múltiplos marcadores simultâneos de variados tipos. A forma mais simples de marcadores oferecidas pela biblioteca são os marcadores *frame*, que são marcadores de identificação única codificados com um padrão binário ao longo das bordas, conforme pode ser observado na Figura 2. No entanto, a biblioteca Vuforia não é restrita somente a estes. Imagens diversas, textos ou imagens escolhidas em tempo de execução podem ser utilizadas como marcadores.

Figura 2 - Marcador do tipo frame

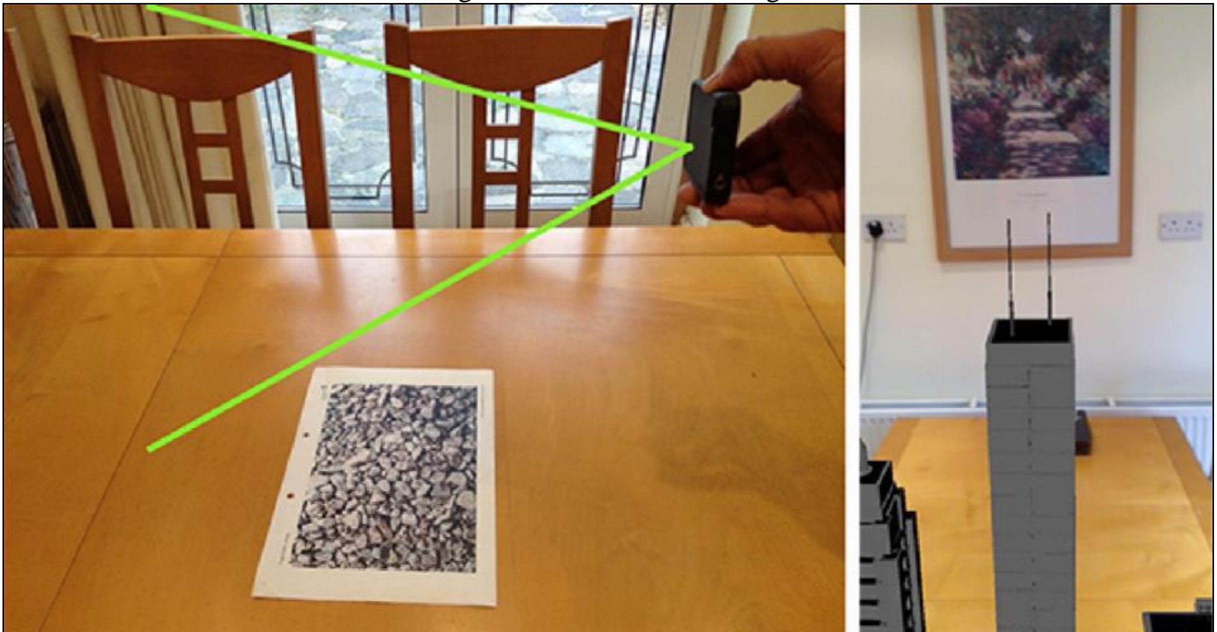



Fonte: QualComm (2015a).

Uma das possibilidades da biblioteca que demonstra seu potencial é o `Extended Tracking`. Segundo a QualComm (2014b), com esta ferramenta é possível manter uma experiência visual contínua mesmo se o alvo que se mantém como base para a Realidade Aumentada não esteja mais sendo visualizado através da câmera, conforme demonstrado na Figura 3.

A biblioteca possui, em seu site, uma grande quantidade de aplicações exemplo, documentação e tutorais além de uma área de suporte, facilitando o aprendizado por parte dos desenvolvedores. Neste mesmo site a ferramenta é disponibilizada com suas funcionalidades básicas gratuitamente, havendo funcionalidades que só podem ser obtidas através da aquisição de uma versão paga.

Figura 3 - Extended Tracking



Fonte: QualComm (2014b).

## 2.5 TRABALHOS CORRELATOS

Existem diversos trabalhos acadêmicos, bibliotecas e aplicativos comerciais que auxiliam, descrevem e ou implementam o processo de criação de um aplicativo de Realidade Aumentada. Dentre esses foram escolhidos o trabalho de Vasselai (2010), além da biblioteca ARToolKit e a *engine* de desenvolvimento de jogos Unreal Engine.

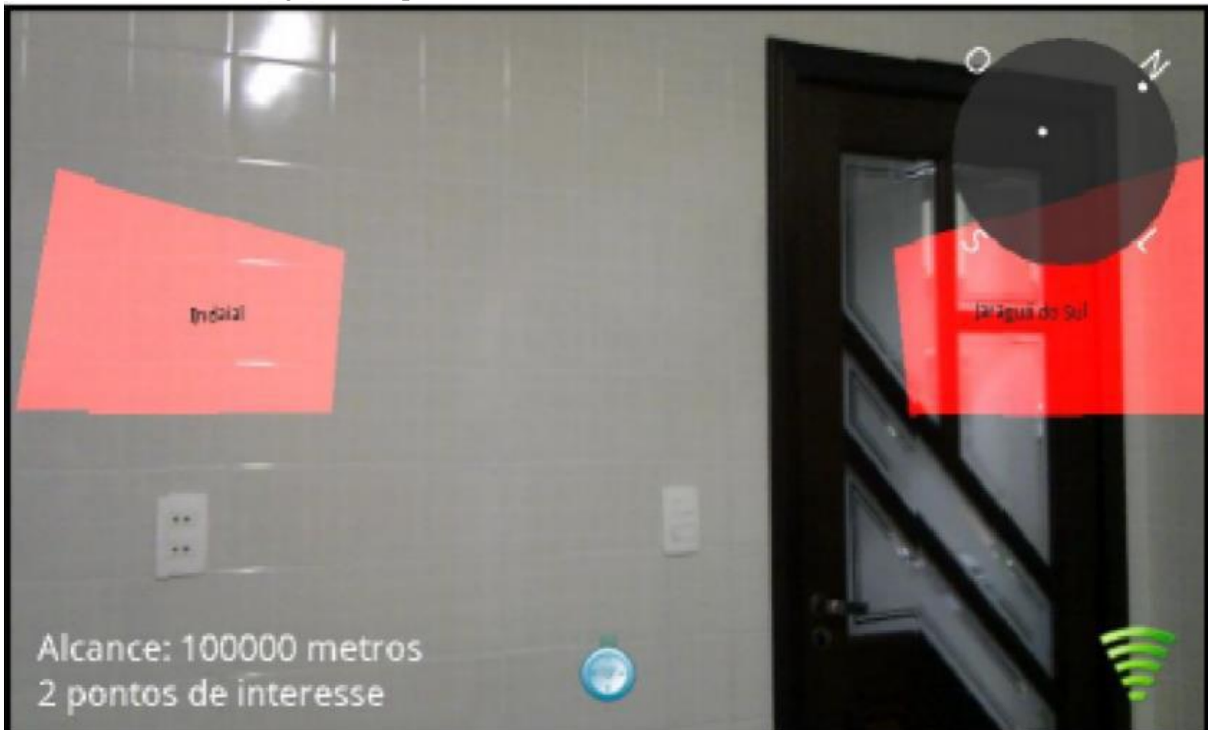
### 2.5.1 Um Estudo Sobre Realidade Aumentada para a Plataforma Android

O trabalho de Vasselai (2010) desenvolveu uma aplicação para Android envolvendo os conceitos de Realidade Aumentada, elucidando o potencial da plataforma frente a este conceito e utilizando os sensores disponíveis na plataforma para uma melhor interação do usuário. Nesse trabalho é produzida uma aplicação que integra a realidade com a virtualidade demonstrando pontos de interesse, baseados em coordenadas geográficas, sobre as imagens obtidas pela câmera de vídeo do aparelho Android, produzindo, assim, a impressão do aumento de realidade.

Desenvolvido em 2010, este trabalhou enfrentou os limites técnicos existentes na época. Um exemplo destes limites técnicos é visto na biblioteca OpenGL ES, que ainda estava em sua versão 1.0. Mesmo com os desafios técnicos da época, o produto do trabalho de Vasselai (2010) consegue atender as três propriedades básicas que definem a Realidade Aumentada segundo Azuma (1997, p. 2). A integração do virtual com o real é percebido com a sobreposição de informações sobre a imagem da câmera, conforme demonstrado na Figura

4. O rastreamento dos objetos em terceira dimensão e a interatividade em tempo real são verificados através da reação dos objetos em cena ao movimento do aparelho, bem como na possibilidade de o usuário selecionar os painéis virtuais da tela exibidos sobre o mundo real.

Figura 4 - Aplicativo de Vasselai (2010) em funcionamento




Fonte: Vasselai (2010).

### 2.5.2 ARToolKit

ARToolKit é uma das mais populares bibliotecas de RA tendo sido baixada mais de 650 mil vezes de seu repositório (DAQRI, 2015c). Distribuída de maneira open source, a biblioteca permite a programadores o desenvolvimento facilitado de aplicações de RA.

Originalmente desenvolvido pelo Dr. Hirokazu Kato, hoje tem a continuação de seu desenvolvimento suportada pelo Human Interface Technology Laboratory (HIT Lab) da universidade de Washington, pelo HIT Lab NZ da Universidade de Canterbury, Nova Zelândia, e pela desenvolvedora DAQRI (LAMB, 2010). Possui suporte às principais plataformas do mercado (Mac OS X, PC, Linux, Android e iOS) além de possuir um plugin para o desenvolvimento com Unity3D (DAQRI, 2015a) sendo possível sua utilização com as linguagens C++, Java (Android), Objective C (iOS, OS X) e C# (DAQRI, 2015b).

A biblioteca tem como características principais a possibilidade de utilizar marcadores de características naturais como uma foto ou imagem comum, suporte a calibração da câmera, rastreamento simultâneo de marcadores, suporte a Unity3D e OpenSceneGraph além de ser otimizado para dispositivos móveis (DAQRI, 2015a).

A biblioteca utiliza técnicas de visão computacional para calcular a posição real da câmera e orientação relativa aos marcadores, sendo que esse processo é realizado em tempo real permitindo que informações virtuais possam ser perfeitamente alinhadas com objetos reais, mantendo assim uma impressão de RA mais imersiva (DAQRI, 2015a).

### 2.5.3 Unreal Engine

A Unreal Engine é uma *engine* de jogos multiplataforma desenvolvida pela empresa Epic Games. A *engine* possibilita o desenvolvimento de uma vasta gama de jogos que vão desde a plataforma *mobile* (iOS e Android) até plataformas com mais poder de processamento como videogames da geração atual (PlayStation 4 e Xbox One) e computadores (Windows, Mac e Linux) (EPICGAMES, 2015c).

O desenvolvimento dos jogos pode ocorrer tanto por programação direta do desenvolvedor, com o uso da linguagem C++, quanto pela utilização de um recurso da *engine* conhecido como Blueprint no qual é possível o desenvolvimento visual de todo o jogo, gráficos e funcionalidades, através de uma IDE própria (EPICGAMES, 2015a).

Sendo desenvolvida em C++ a *engine* permite o acesso e a alteração de seu código fonte pelos desenvolvedores de maneira que é possível entender ou customizar funções que vão desde as do editor até os subsistemas incluídos na mesma como o de física, áudio, comunicação, animação, renderização, entre outros. Tal possibilidade de customização, além da ampla documentação, atrai a comunidade de desenvolvedores de modo que somente no serviço de compartilhamento de projetos GitHub existe aproximadamente três milhões de linhas de código de projetos abertos voltados à *engine* (EPICGAMES, 2015c). Além dos projetos abertos a ferramenta oferece o serviço Marketplace, no qual desenvolvedores conseguem oferecer comercialmente recursos desenvolvidos como artes, efeitos sonoros ou scripts.

A empresa desenvolvedora oferece a *engine* completa de maneira gratuita, incluindo o código fonte, mas com o acordo que o desenvolvedor pague 5% da receita bruta após os primeiros US$ 3000 por produto por trimestre (EPICGAMES, 2015b).

# 3 DESENVOLVIMENTO DA BIBLIOTECA

Este capítulo detalha as etapas do desenvolvimento da biblioteca. São apresentados os requisitos, a especificação e a implementação da mesma, mencionando as técnicas e tecnologias utilizadas. Também são comentadas questões referentes à operacionalidade da biblioteca e os resultados obtidos.

## 3.1 REQUISITOS PRINCIPAIS DA BIBLIOTECA A SER DESENVOLVIDA

A biblioteca proposta deve atender os seguintes requisitos:

a) permitir visualizar o ambiente real utilizando a câmera de vídeo de um dispositivo Android (Requisito Funcional – RF);
b) permitir identificar marcadores binários simples e palavras definidas pelo usuário de forma a serem utilizados como pontos de embasamento de localização espacial para a Realidade Aumentada (RF);
c) permitir sobrepor objetos virtuais ao ambiente real (RF);
d) permitir notificações, em tempo real, entre os dispositivos rodando a ferramenta na mesma rede local (RF);
e) permitir o carregamento e reprodução de arquivos de áudio (RF);
f) ser implementada sobre a plataforma Android (Requisito Não Funcional – RNF);
g) utilizar a biblioteca OpenGL ES (RNF);
h) permitir o uso de arquivos de áudio nos formatos suportados pela plataforma Android (RNF);
i) ser especificada utilizando-se de análise orientada a objetos com *Unified Modeling Language* (UML) na ferramenta Enterprise Architecht versão 10.0 (RNF).

## 3.2 ESPECIFICAÇÃO

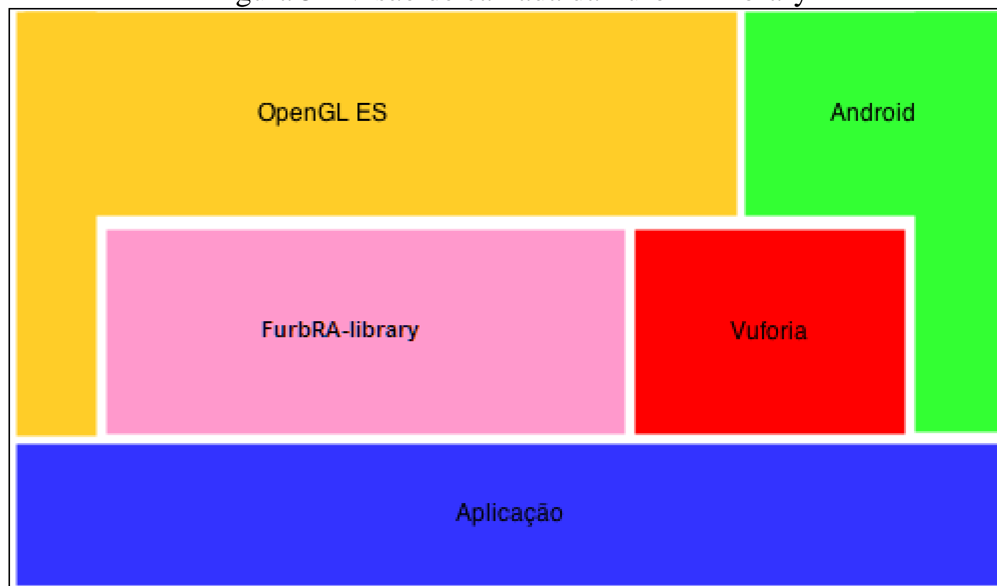
A especificação da biblioteca em questão foi desenvolvida seguindo a análise orientada a objetos, utilizando a notação *Unified Modeling Language* (UML) em conjunto com a ferramenta Enterprise Architect 12.0.1214 para a elaboração dos casos de uso e dos diagramas de classes.

### 3.2.1 Visão da biblioteca

A biblioteca de realidade aumentada FurbRA-library é uma camada intermediária que agrupa quatro áreas necessárias a uma biblioteca de RA para jogos multijogadores: renderização gráfica, RA, comunicação e multimídia. A biblioteca se localiza entre a

aplicação, a biblioteca OpenGL ES (utilizada como biblioteca gráfica), a biblioteca Vuforia (utilizado para a criação da RA) e o *framework* Android (utilizado para comunicação e multimídia) como pode ser observado na Figura 5. Apesar da biblioteca disponibilizar rotinas para o desenvolvimento da aplicação, ainda é possível que o desenvolvedor acesse diretamente os recursos disponíveis nas bibliotecas Vuforia e OpenGL ES e no *framework* Android.

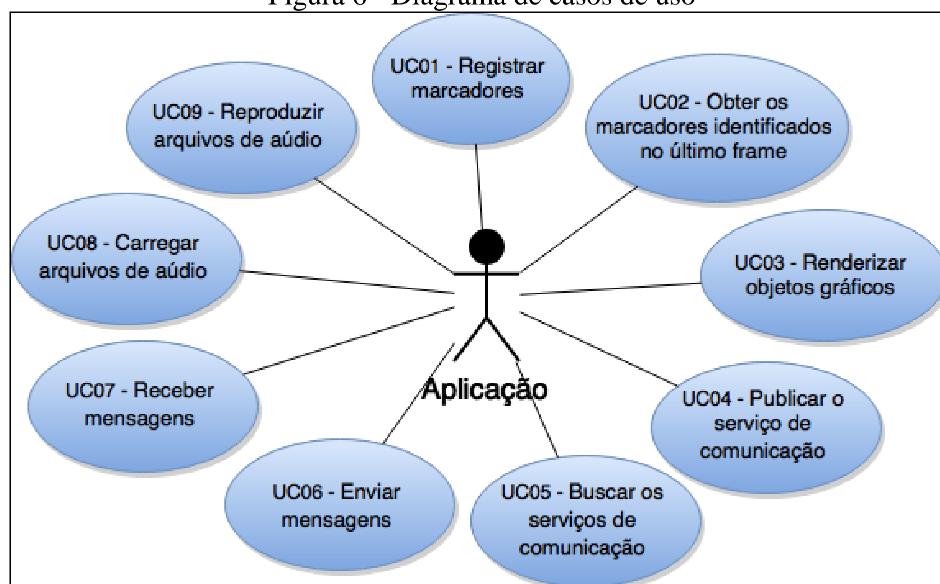
Figura 5 - Visão de camada da FurbRA-library




### 3.2.2 Casos de Uso

Nesta seção são descritos os casos de uso da FurbRA-library. Foi identificado como sendo o ator principal a `Aplicação`. Na Figura 6 é apresentado o diagrama de casos de uso da biblioteca FurbRA-library.

Figura 6 - Diagrama de casos de uso



O caso de uso UC01 - `Registrar marcadores` descreve o processo de registro de marcadores de modo que estes possam ser rastreados e obtidos a partir da imagem da câmera. Detalhes deste caso de uso estão descritos no Quadro 1.

Quadro 1 - Caso de uso UC01

| UC01 - Registrar marcadores | |
|---|---|
| Descrição | Permitir configurar marcadores de maneira que possam ser identificados na imagem da câmera. |
| Cenário Principal | 1. A `Aplicação` informa à biblioteca os tipos de marcadores que serão utilizados.<br>2. A `Aplicação` informa à biblioteca os dados dos marcadores que serão utilizados. |
| Pós-Condição | A biblioteca possui as configurações necessárias para obter os marcadores na imagem da câmera. |

O caso de uso UC02 - `Obter os marcadores identificados no último frame` descreve o processo para obtenção dos marcadores identificados. Detalhes deste caso de uso estão descritos no Quadro 2.

Quadro 2 - Caso de uso UC02

| UC02 – Obter os marcadores identificados no último frame | |
|---|---|
| Descrição | Permitir obter os marcadores, configurados segundo o caso de uso UC01, a partir da imagem da câmera de maneira que possam ser utilizados como base para a RA. |
| Cenário Principal | 1. A biblioteca obtém os marcadores identificados.<br>2. A biblioteca repassa à `Aplicação` os marcadores identificados. |
| Pós-Condição | A `Aplicação` recebe notificações quanto os marcadores, previamente configurados, identificados na imagem da câmera. |

O caso de uso UC03 - `Renderizar objetos gráficos` descreve o processo de desenho na camada de renderização. Detalhes deste caso de uso estão descritos no Quadro 3.

Quadro 3 - Caso de uso UC03

| UC03 – Renderizar objetos gráficos | |
|---|---|
| Descrição | Permitir renderizar objetos gráficos. |
| Cenário Principal | 1. A biblioteca solicita à `Aplicação` a preparação dos objetos gráficos no início da cena.<br>2. A biblioteca solicita à `Aplicação` a renderização dos objetos gráficos a cada *frame*.<br>3. A biblioteca solicita à `Aplicação` a finalização dos objetos gráficos renderizados a cada *frame*. |
| Pós-Condição | Os objetos gráficos são renderizados. |

O caso de uso UC04 - `Publicar o serviço de comunicação` descreve o processo de publicação do serviço de comunicação. Detalhes deste caso de uso estão descritos no Quadro 4.

Quadro 4 - Caso de uso UC04

| UC04 – Publicar o serviço de comunicação | |
|---|---|
| Descrição | Permitir que os aplicativos possam se encontrar entre si de forma a possibilitar uma comunicação. |
| Pré-Condição | O dispositivo necessita possuir uma conexão WiFi ativa. |
| Cenário Principal | 1. A `Aplicação` solicita a publicação do serviço de comunicação.<br>2. A biblioteca obtém a porta de comunicação que o serviço irá divulgar.<br>3. A biblioteca obtém o nome do serviço que será divulgado.<br>4. A biblioteca publica na rede local o serviço de comunicação. |
| Pós-Condição | O serviço de comunicação é publicado na rede local. |

O caso de uso UC05 - `Buscar os serviços de comunicação` descreve o processo de obtenção dos serviços de comunicação das instâncias do aplicativo na rede local. Detalhes deste caso de uso estão descritos no Quadro 5.

Quadro 5 - Caso de uso UC05

| UC05 – Buscar os serviços de comunicação | |
|---|---|
| Descrição | Permitir que a `Aplicação` possa encontrar outras instâncias do aplicativo na rede local de forma a permitir a comunicação. |
| Pré-Condição | O dispositivo necessita possuir uma conexão WiFi ativa e necessita ter publicado o serviço de comunicação conforme o caso de uso UC05. |
| Cenário Principal | 1. A `Aplicação` solicita à biblioteca a descoberta de serviços na rede local.<br>2. A cada serviço descoberto à biblioteca atualiza a lista de serviços conhecidos.<br>3. A biblioteca informe à `Aplicação` quanto ao novo serviço descoberto.<br>4. A `Aplicação` solicita à biblioteca os serviços conhecidos. |
| Pós-Condição | Os serviços publicados na rede local são obtidos pela `Aplicação`. |

O caso de uso UC06 - `Enviar mensagens` descreve o processo de envio de mensagens a outro dispositivo rodando o aplicativo na rede local. Detalhes deste caso de uso estão descritos no Quadro 6.

Quadro 6 - Caso de uso UC06

| UC06 – Enviar mensagens | |
|---|---|
| Descrição | Permitir que a `Aplicação` possa enviar mensagens a outra instância do aplicativo rodando na rede local. |
| Pré-Condição | O dispositivo necessita possuir uma conexão WiFi ativa, necessita ter publicado o serviço de comunicação conforme o caso de uso UC04 e realizado a busca dos serviços existentes conforme o caso de uso UC05. |
| Cenário Principal | 1. A `Aplicação` solicita os serviços conhecidos à biblioteca.<br>2. A `Aplicação` escolhe entre os serviços conhecidos qual serviço será o destinatário da mensagem.<br>3. A `Aplicação` cria a mensagem.<br>4. A `Aplicação` solicita à biblioteca o envio da mensagem criada ao serviço destinatário escolhido. |
| Pós-Condição | A mensagem é enviada ao serviço destinatário. |

O caso de uso UC07 - `Receber mensagens` descreve o processo de notificação de recebimento de mensagens de outro aplicativo da rede local. Detalhes deste caso de uso estão descritos no Quadro 7.

Quadro 7 - Caso de uso UC07

| UC07 – Receber mensagens | |
|---|---|
| Descrição | Permitir que a `Aplicação` possa ser notificado quanto ao recebimento de mensagens de outro aplicativo da rede local. |
| Pré-Condição | O dispositivo necessita possuir uma conexão WiFi ativa, necessita ter publicado o serviço conforme o caso de uso UC04. |
| Cenário Principal | 1. A `Aplicação` registra com a biblioteca o interesse quanto ao recebimento de mensagens.<br>2. A biblioteca recebe uma mensagem.<br>3. A biblioteca notifica à `Aplicação` quanto ao recebimento da mensagem.<br>4. A biblioteca entrega à `Aplicação` a mensagem recebida. |
| Pós-Condição | A `Aplicação` é notificada quanto ao recebimento da mensagem. |

O caso de uso UC08 - `Carregar arquivos de áudio` descreve o processo de carregamento prévio dos arquivos de áudio que serão utilizados pela aplicação. Detalhes deste caso de uso estão descritos no Quadro 8.

Quadro 8 - Caso de uso UC08

| UC08 – Carregar arquivos de áudio | |
|---|---|
| Descrição | Permitir que a `Aplicação` possa carregar arquivos de áudio. |
| Cenário Principal | 1. A `Aplicação` registra com a biblioteca os arquivos que devem ser carregados juntamente a um identificador único para cada arquivo.<br>2. A `Aplicação` solicita o carregamento dos arquivos.<br>3. A biblioteca notifica à `Aplicação` quanto à finalização do carregamento dos arquivos. |
| Pós-Condição | A `Aplicação` é notificada quanto à finalização do carregamento dos arquivos e possível utilização dos mesmos. |

O caso de uso UC09 - `Reproduzir arquivos de áudio` descreve o processo de reprodução dos arquivos de áudio. Detalhes deste caso de uso estão descritos no Quadro 9.

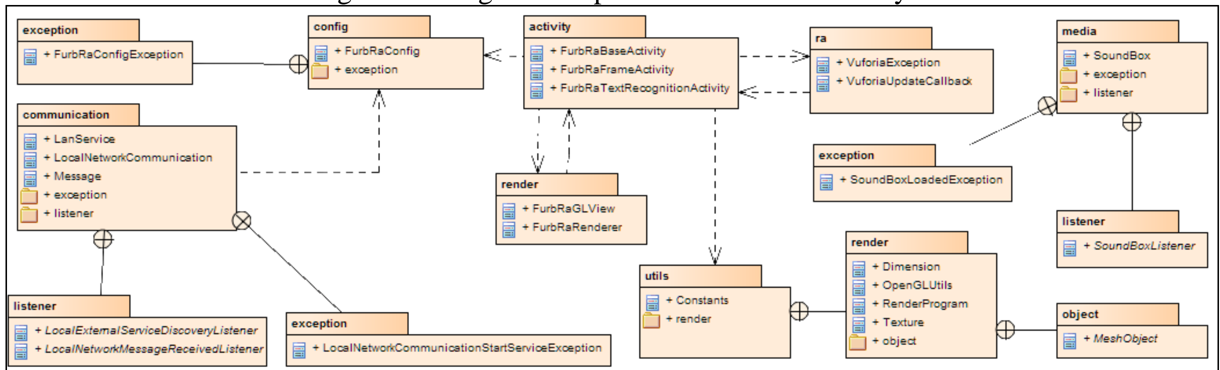
Quadro 9 - Caso de uso UC09

| UC09 – Reproduzir arquivos de áudio | |
|---|---|
| Descrição | Permitir que a `Aplicação` possa reproduzir arquivos de áudio. |
| Pré-Condição | A `Aplicação` necessita ter realizado o carregamento dos arquivos previamente conforme o caso de uso UC08. |
| Cenário Principal | 1. A `Aplicação` solicita a biblioteca a reprodução de um arquivo sonoro informando o identificador único desse arquivo.<br>2. A biblioteca reproduz o arquivo de áudio. |
| Pós-Condição | O arquivo de áudio é reproduzido no dispositivo. |

### 3.2.3 Diagrama de Classes

Nesta seção são descritas as classes e estruturas desenvolvidas que formam toda a FurbRA-library. Para facilitar o entendimento de como as classes estão reunidas, na Figura 7 estão sendo demonstrados os pacotes, suas dependências bem como as classes que os compõem.

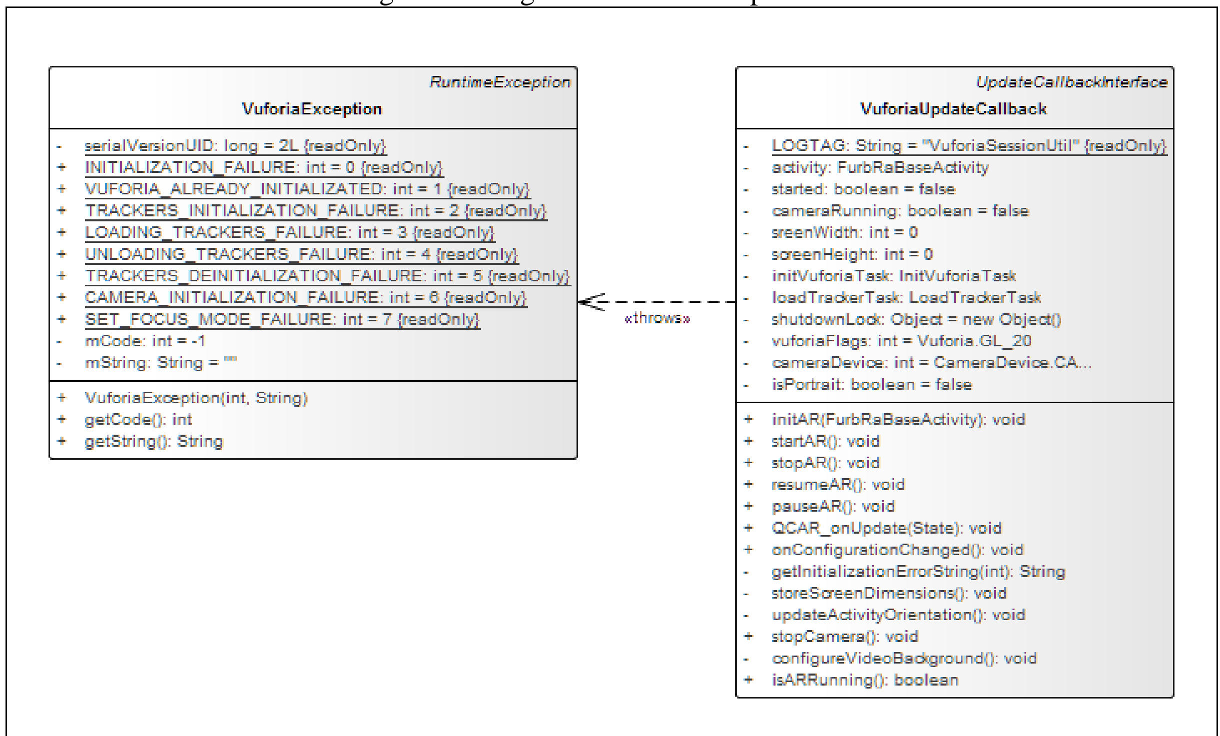
Figura 7 - Diagrama de pacotes da FurbRA-library




### 3.2.3.1 Pacote `ra`

O primeiro pacote é denominado `ra` e abriga as classes relacionadas a comunicação com a biblioteca Vuforia, conforme pode ser observado na Figura 8.

Figura 8 - Diagrama de classe do pacote `ra`

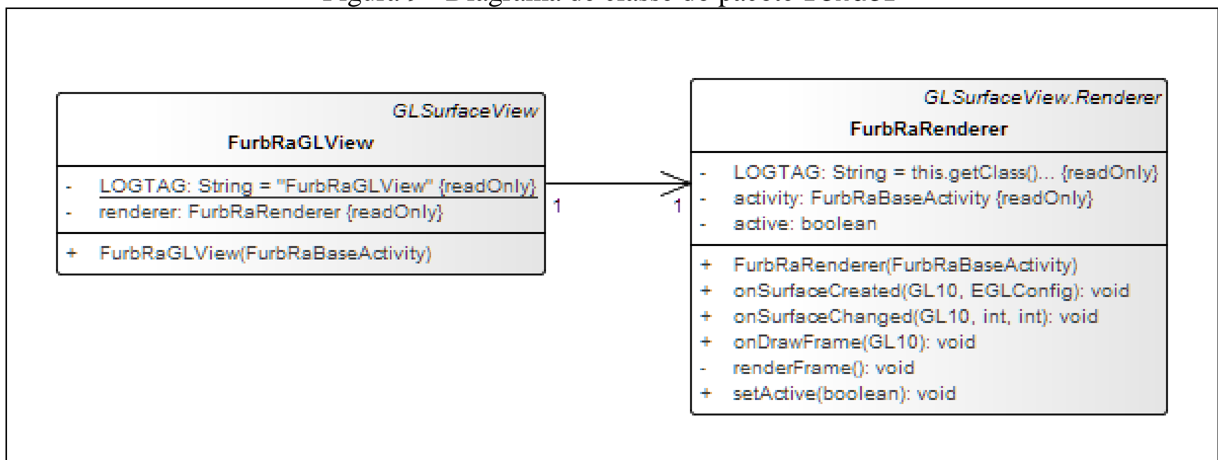



A classe `VuforiaUpdateCallback` é a principal classe desse pacote. Esta implementa a interface `UpdateCallbackInterface` oferecida pela biblioteca Vuforia e possui a responsabilidade do gerenciamento do ciclo de execução da mesma. A classe `VuforiaException` representa uma falha na execução gerada durante as fases da biblioteca Vuforia.

#### 3.2.3.2 Pacote `render`

O pacote denominado `render` (Figura 9) possui as classes responsáveis por renderizar a camada OpenGL ES.

Figura 9 - Diagrama de classe do pacote `render`



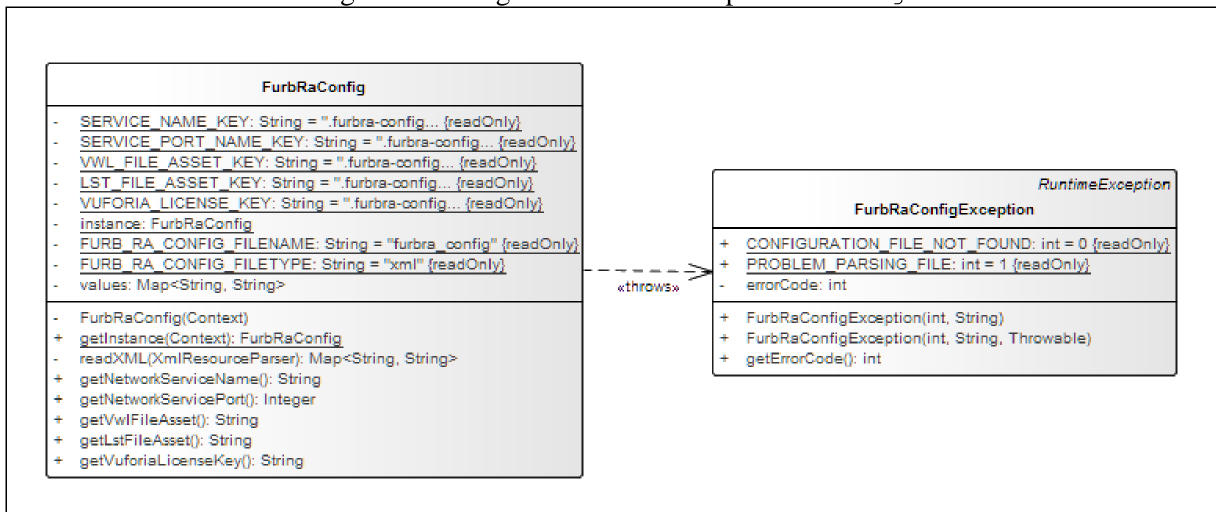

Toda renderização OpenGL ES no *framework* Android necessita ocorrer através da classe `GLSurfaceView`. Esta classe permite a configuração necessária a renderização delegando a ação de fato à classe `GLSurfaceView.Renderer`. Na biblioteca FurbRA-library estas classes foram especializadas criando-se, respectivamente, as classes `FurbRaGLView` e `FurbRaRenderer`. A classe mais importante de ser observada nesse pacote é a `FurbRaRenderer` já que é nesta classe, mais especificamente no método `onDrawFrame`, que as principais ações envolvendo a RA irão ocorrer. Neste método são realizadas as seguintes ações:

a) obtenção e renderização da imagem da câmera;
b) delegação quanto a criação de objetos gráficos em uma camada superior a imagem da câmera;
c) comunicação com a biblioteca Vuforia de modo a obter os marcadores identificados no último frame.

#### 3.2.3.3 Pacote `config`

O pacote denominado `config` (Figura 10) possui as classes responsáveis por obter, processar e fornecer as informações inseridas pelo desenvolvedor no arquivo de configurações da biblioteca denominado furbra_config.xml.

Figura 10 - Diagrama de classe do pacote `config`



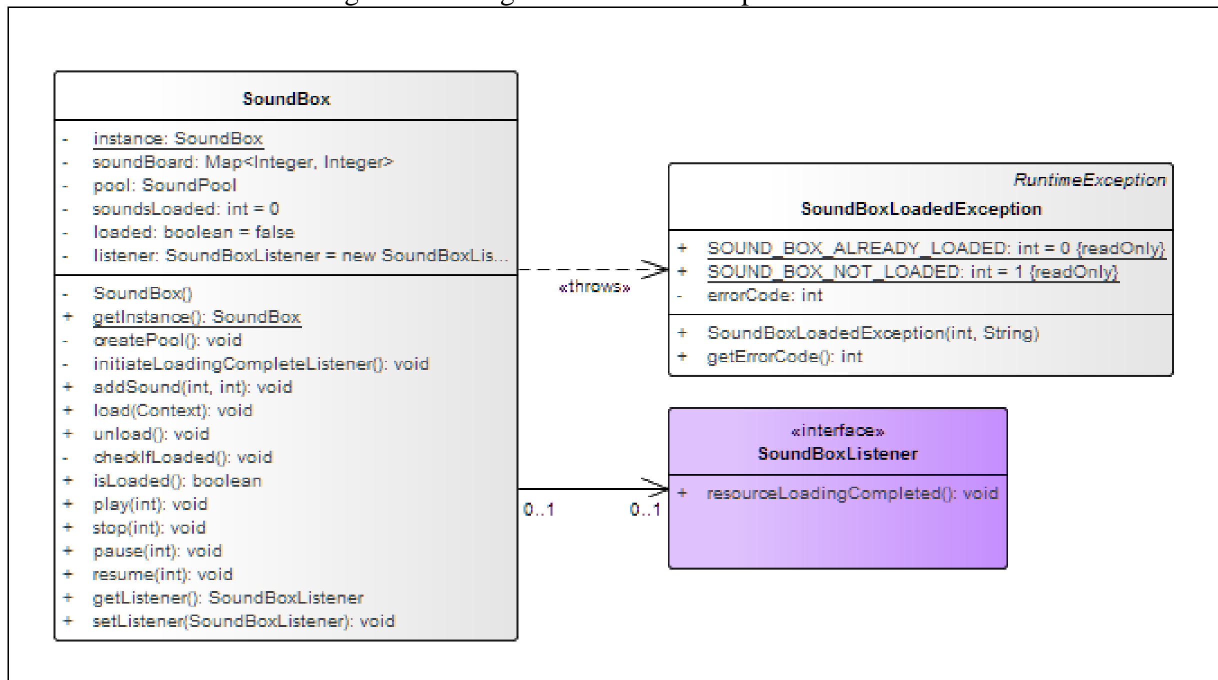

A principal classe desse pacote, `FurbRaConfig`, é estruturada seguindo o padrão de projeto *singleton,* de modo que toda a aplicação compartilhe da mesma instância do arquivo de configuração. O arquivo de configuração é obtido somente no método construtor, chamado unicamente na primeira chamada da aplicação ao método `getInstance`. Durante a execução deste construtor o sistema pode lançar a `exception FurbRaConfigException`, existente no pacote `exception`, de modo a sinalizar uma de duas seguintes possibilidades:

a) o arquivo não foi encontrado, o que fará a biblioteca lançar a `exception FurbRaConfigException` que possui o código de erro definido na variável estática `FurbRaConfigException.CONFIGURATION_FILE_NOT_FOUND`;
b) o arquivo não está propriamente formatado, o que fará a biblioteca lançar a `exception FurbRaConfigException` que possui o código de erro definido na variável estática `FurbRaConfigException.PROBLEM_PARSING_FILE`.

#### 3.2.3.4 Pacote `media`

O pacote denominado `media` (Figura 11) possui as classes responsáveis por permitir recursos de mídia às aplicações. A implementação atual conta com recursos para utilização de mídias sonoras.

Figura 11 - Diagrama de classes do pacote `media`



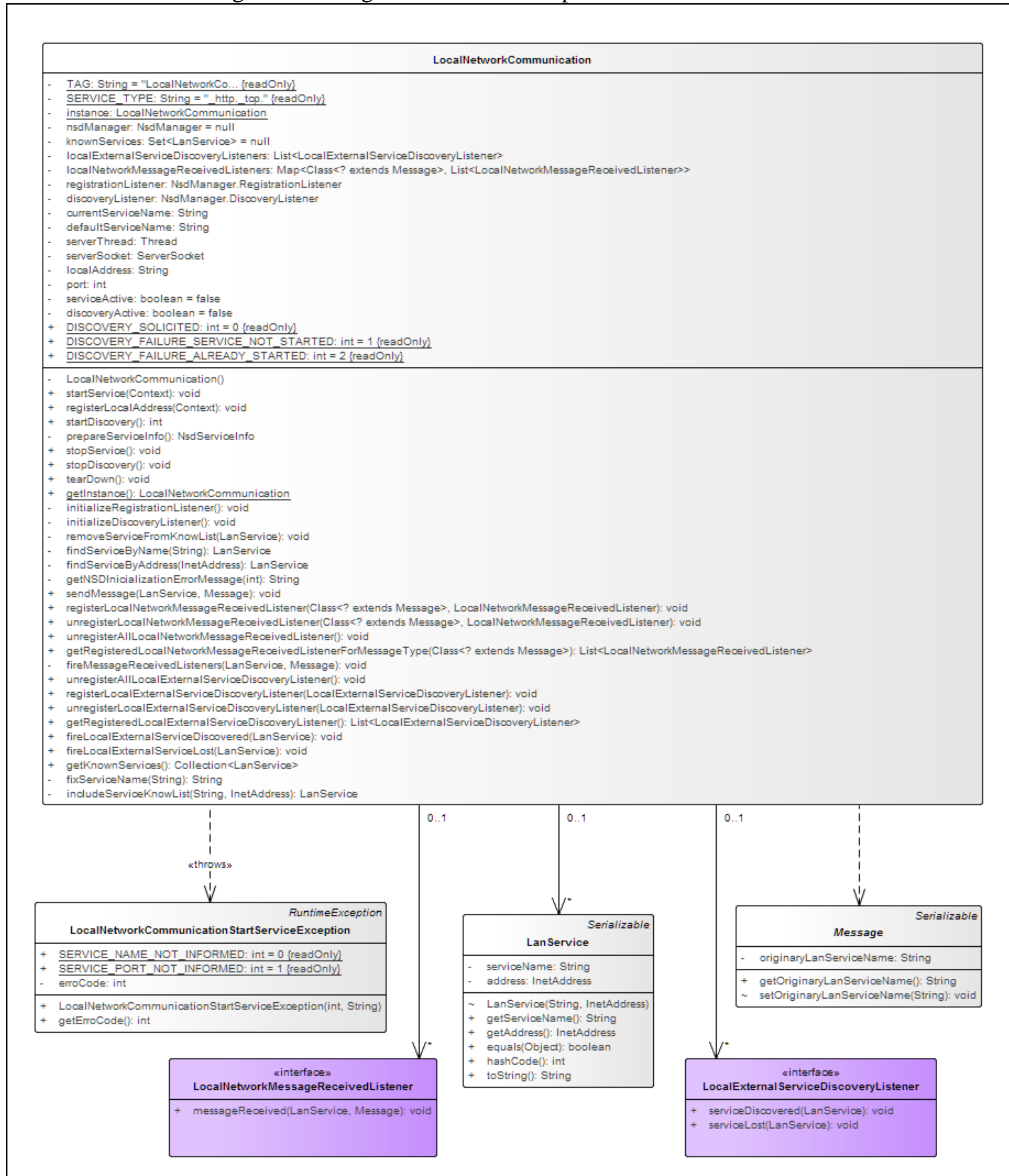

A classe `SoundBox`, principal classe do pacote, é estruturada seguindo o padrão de projeto *singleton* e possui a responsabilidade de carregar e gerenciar os recursos de som da aplicação. Esta classe realiza o carregamento assíncrono dos recursos de modo que a *Thread* principal não seja sobrecarregada. A implementação da classe `SoundBox` permite que o mesmo recurso de áudio não seja carregado mais de uma vez na memória do dispositivo, fazendo esse processo de controle de forma transparente. Como o carregamento é assíncrono necessita-se de uma metodologia de eventos para sinalizar a finalização do processo de carregamento e possível utilização dos recursos. Este evento é definido através da `interface SoundBoxListener`. A classe `SoundBoxLoadException`, por sua vez representa um problema na utilização da classe `SoundBox` em detrimento ao atual estado de carregamento dos recursos, de modo que o seu lançamento, pela biblioteca, indica uma das seguintes possibilidades:

a) os recursos estão sendo solicitados, através de métodos de manipulação dos mesmos como `play`, `pause`, etc, sem que o carregamento dos recursos tenha sido realizado ou finalizado;
b) o método `load`, destinado ao carregamento dos recursos, está sendo utilizado pela aplicação após o processo de carregamento dos recursos já ter sido solicitado.

#### 3.2.3.5 Pacote `communication`

O pacote denominado `communication` (Figura 12) possui as classes responsáveis por permitir o registro, descoberta e comunicação dos aplicativos desenvolvidos com a biblioteca FurbRA-library dentro da rede local.

Figura 12 - Diagrama de classes do pacote `communication`




A classe `Message` define uma mensagem trafegada entre as aplicações. A classe possui como atributo o nome do serviço originário da mesma. A classe `LanService` define os

serviços locais encontrados pela aplicação, possuindo como atributos as informações quanto a esse serviço na rede como o nome e endereço. A classe `LocalNetworkCommunication` é a principal classe do pacote, estruturada seguindo o padrão de projeto *singleton,* e possui a responsabilidade de gerenciar toda o processo de comunicação local. A classe utiliza o recurso NSD, do *framework* Android, para publicar e descobrir instâncias do mesmo aplicativo rodando na rede local. A comunicação entre os serviços é realizada assincronamente de modo que a classe `LocalNetworkCommunication` possui métodos para o envio e para notificações quanto ao recebimento de mensagens, sendo que em nenhum dos dois casos o processo ocorre na `Thread` principal da aplicação.

A interface `LocalExternalServiceDiscoveryListener` é utilizada para o recebimento de eventos quanto à descoberta de serviços. Classes que implementam esta interface, e são registradas junto a classe `LocalNetworkCommunication`, são notificadas quanto a descobertas de novos serviços ou quanto a perda de comunicação com os mesmos. A interface `LocalNetworkMessageReceivedListener` é utilizada para o recebimento de notificações quanto ao recebimento de novas mensagens. Classes com esta característica devem, além de implementar a interface, se registrar junto à classe `LocalNetworkMessageReceivedListener` através do método `registerLocalNetworkMessageReceivedListener`, passando como parâmetro a este método a classe que implementa a interface e o tipo de mensagem que a classe deseja notificação quanto ao recebimento.

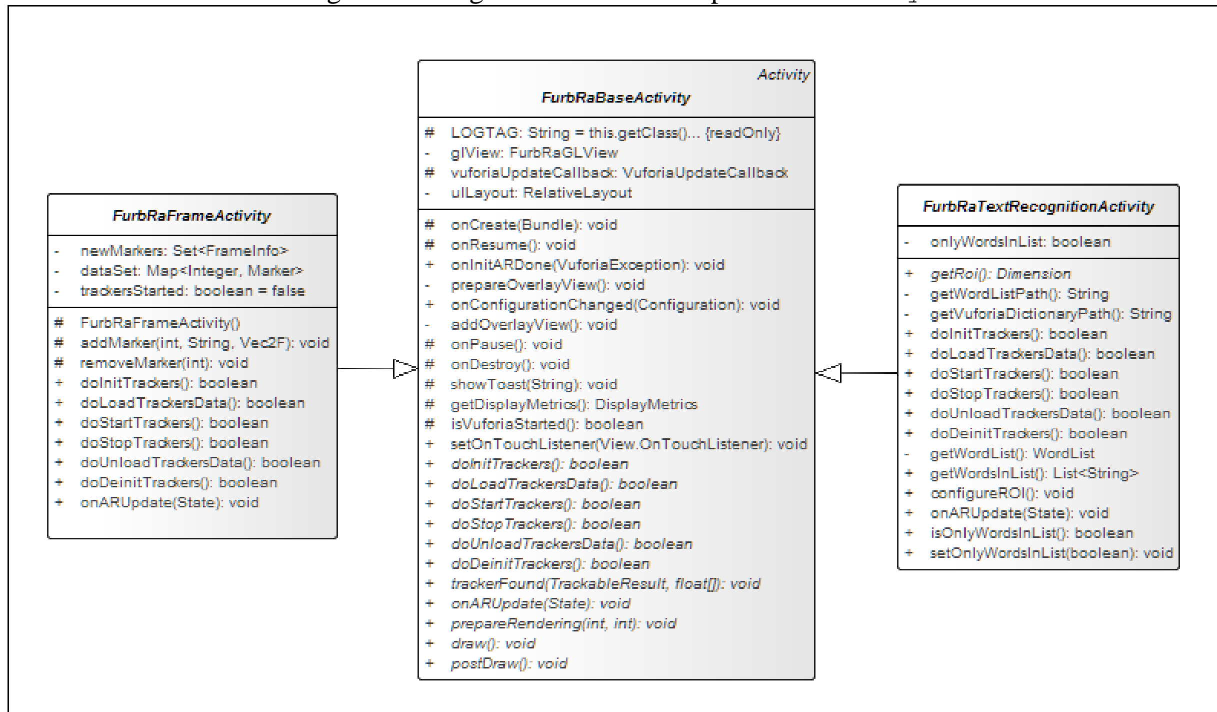
A classe `LocalNetworkCommunicationStartServiceException`, existente no pacote `exception`, pode ser lançada pela biblioteca, de modo a sinalizar uma de duas seguintes possibilidades:

a) o arquivo `furbra-config.xml` não contém a informação quanto ao nome do serviço NSD que deve ser publicado, o que fará a biblioteca lançar a `exception` com o código de erro definido na variável estática `LocalNetworkCommunicationStartServiceException.SERVICE_NAME_NOT_INFORMED`;
b) o arquivo `furbra-config.xml` não contém a informação quanto à porta de comunicação que o serviço NSD deve publicar, o que fará a biblioteca lançar a `exception` com o código de erro definido na variável estática `LocalNetworkCommunicationStartServiceException.SERVICE_PORT_NOT_INFORMED`.

#### 3.2.3.6 Pacote `activity`

O pacote denominado `activity` (Figura 13) possui as classes base para a definição da interface de usuário das aplicações de RA.

Figura 13 - Digrama de classes do pacote `activity`



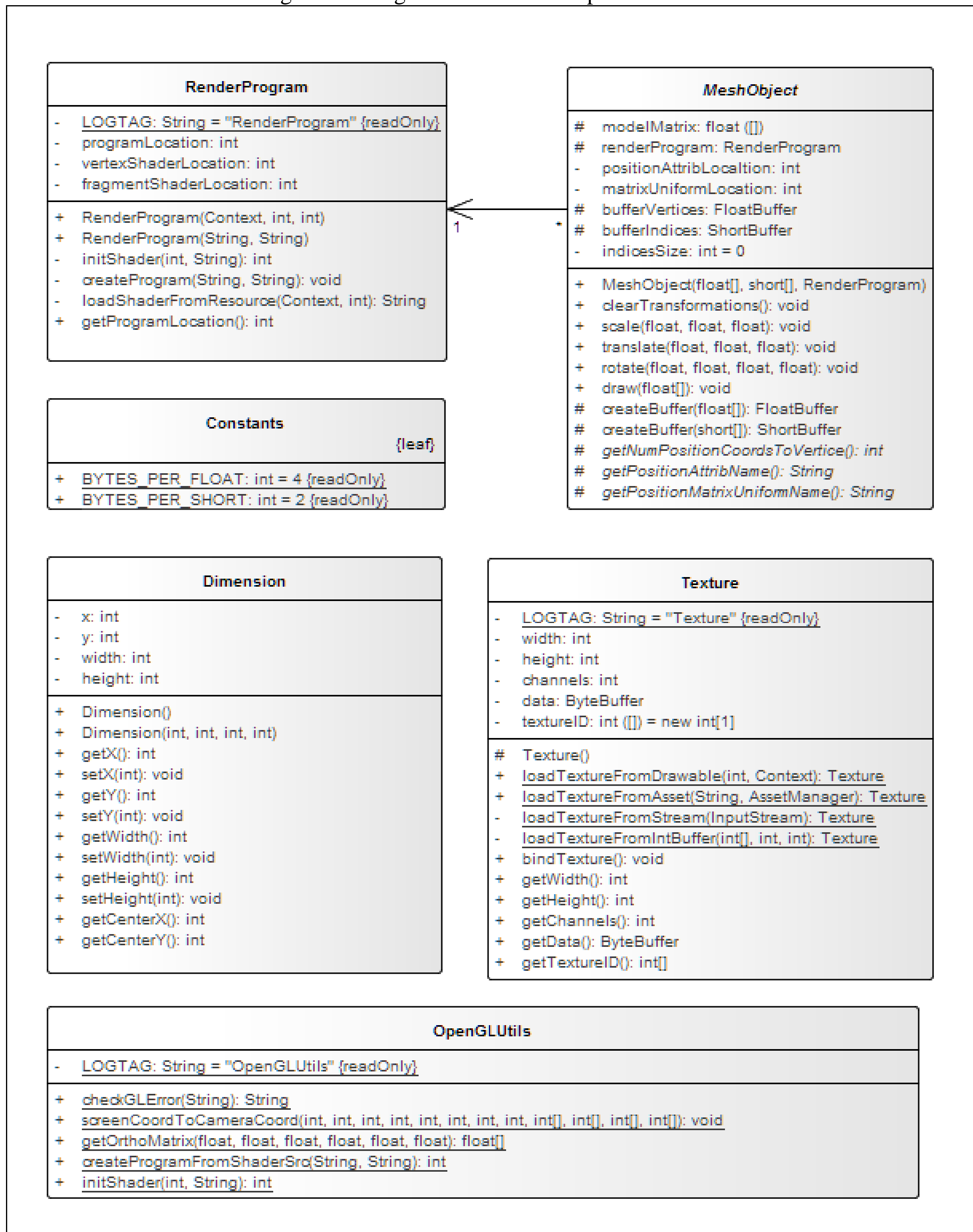

A principal classe do pacote é a `FurbRaBaseActivity`. Esta classe une as classes responsáveis pelo RA, render e interface de usuário, além de definir o fluxo de rotinas que devem ser implementadas e utilizadas de modo que seja possível realizar o carregamento da biblioteca Vuforia, e as informações necessárias a esta, de forma transparente ao desenvolvedor. Esta classe abstrata é moldada de forma que as classes que a estendam sejam responsáveis pelo carregamento e detecção de um tipo especifico de marcador. A classe `FurbRaBaseActivity` é especializada nas seguintes classes:

a) `FurbRaFrameActivity`: classe especializada na detecção de marcadores do tipo `Frame` a partir da imagem da câmera;
b) `FurbRaTextRecognitionActivity`: classe especializada na detecção de palavras pré-definidas a partir da imagem da câmera.

#### 3.2.3.7 Pacote `utils`

O pacote denominado `utils` (Figura 14) possui as classes utilitárias da biblioteca que tem como objetivo facilitar determinadas atividades do usuário.

Figura 14 - Digrama de classes do pacote `utils`




A classe `Constants` consiste em variáveis de uso comum e imutáveis. A classe `Dimension` consiste em um POJO que armazena informações em relação a tamanho e posicionamento, em duas dimensões, sendo ambas associadas entre si. A classe `MeshObject` possui métodos e estruturas voltadas a facilitar a criação de objetos gráficos à serem renderizados pela camada OpenGL ES. A classe `OpenGLUtils` é um combinado de métodos

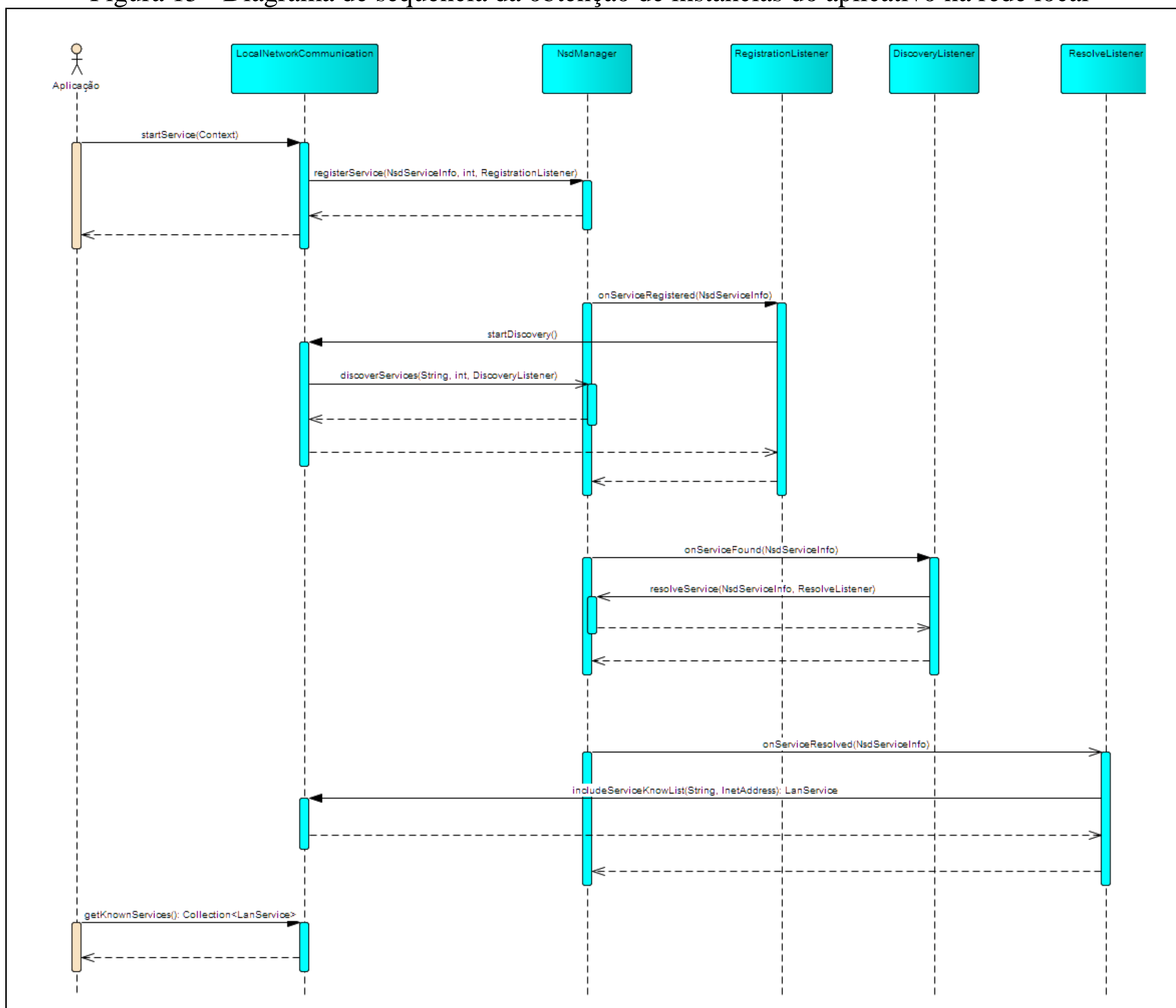
utilitários voltados a utilização das chamadas à API OpenGL ES. A classe `RenderObject` representa um programa OpenGL que possui um `vertex shader` e um `fragment shader`. A classe possui métodos facilitadores à compilação e a criação do programa OpenGL. `Texture` é uma classe que, além de representar uma textura gráfica, possui suporte a importação de dados de imagens e utilização dessas como texturas pelo OpenGL.

### 3.2.4 Diagrama de sequência

O diagrama de sequência da Figura 15 apresenta a interação da `Aplicação` com a biblioteca FurbRA-library na obtenção de instâncias do aplicativo na rede local, através do serviço NSD, processo que ocorre na execução das ações realizadas nos casos de uso UC05 e UC06.

Detalhes sobre o processo realizado são descritos na secção 3.3.9.

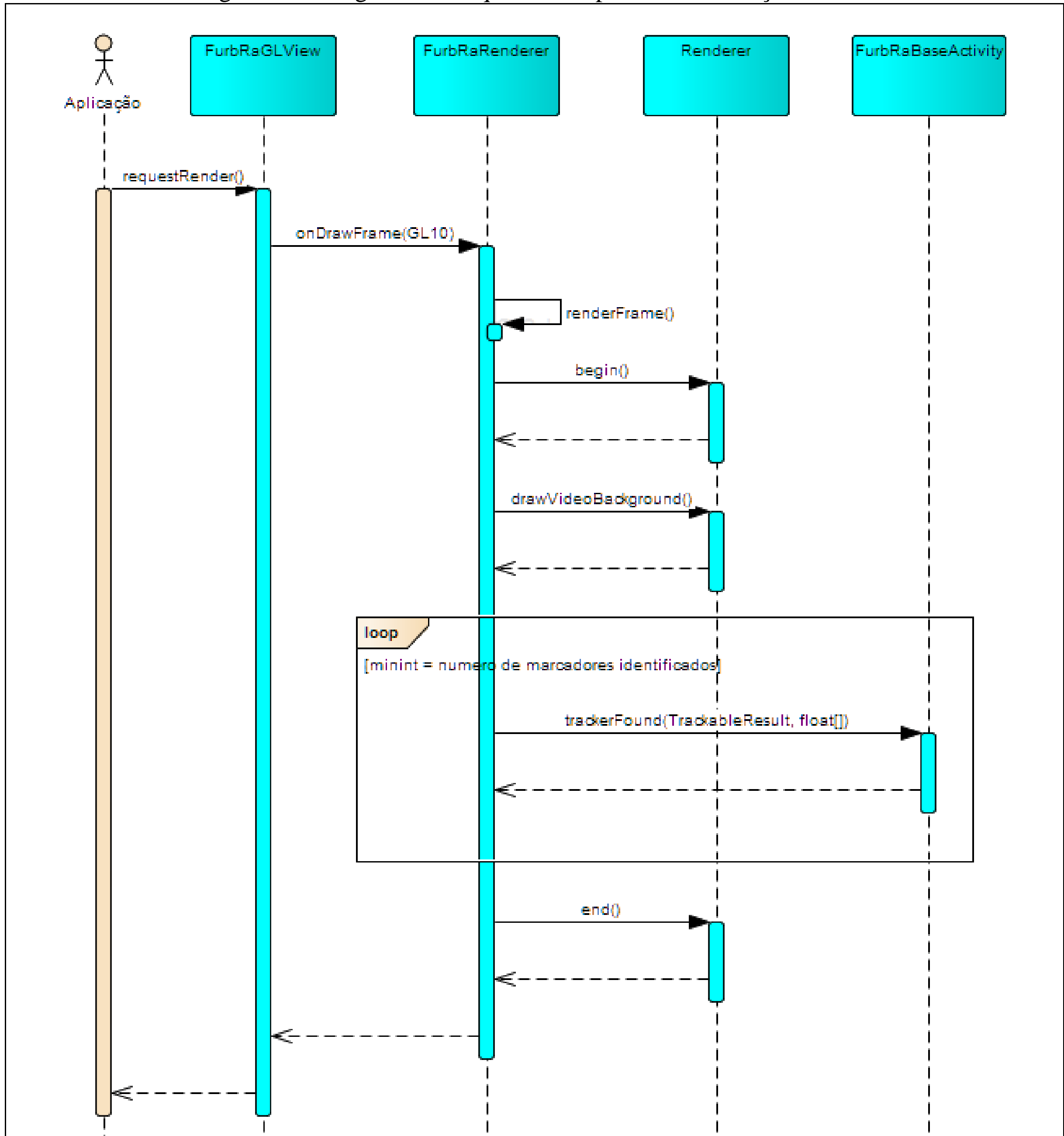
Figura 15 - Diagrama de sequência da obtenção de instâncias do aplicativo na rede local




O diagrama de sequência da Figura 16 apresenta a criação da RA pela biblioteca FurbRA-library, processo que ocorre na execução das ações realizadas no caso de uso UC02.

Detalhes sobre o processo realizado são descritos na secção 3.3.7.

Figura 16 - Diagrama de sequência do processo de criação da RA




## 3.3 IMPLEMENTAÇÃO

A seguir são descritas as técnicas e ferramentas utilizadas para o desenvolvimento da biblioteca proposta, junto com uma descrição da sua operacionalidade e trechos de código para um melhor entendimento.

### 3.3.1 Técnicas e ferramentas utilizadas

Para a implementação da FurbRA-library foi utilizada a linguagem de programação Java, na versão 7. O ambiente de desenvolvimento utilizado foi o Android Studio 1.2.1.1,

disponibilizado pela Google. Para a criação da RA foi utilizado a biblioteca Vuforia na versão 4.0.105. A geração de imagens gráficas no espaço 3D foi implementada utilizando a API OpenGL ES 2.0, desenvolvida para sistemas embarcados.

Para execução, testes e depuração de todo o desenvolvimento do trabalho, utilizou-se dois tablets Android, sendo um tablet Asus Nexus 7 (2012) com sistema operacional Android 5.0 e um tablet Samsung Galaxy Note 10.1 com sistema operacional Android 4.3.

### 3.3.2 Configurações da biblioteca

A classe `FurbRaConfig` é estruturada seguindo o padrão de projeto s*ingleton* e possui como função principal a obtenção do arquivo de configurações da biblioteca FurbRA-library chamado `furbra-config.xml`. A estrutura da classe permite que o arquivo seja lido somente na instanciação da classe (Quadro 10), em sua primeira utilização. A próxima chamada acessará um cache em memória armazenado na primeira chamada.

Quadro 10- Implementação do método construtor da classe `FurbRaConfig`

```
private FurbRaConfig(Context context) {
    int identifier = context.getResources().getIdentifier(
                          FURB_RA_CONFIG_FILENAME, FURB_RA_CONFIG_FILETYPE,
                          context.getPackageName());

    //Se não foi achado o arquivo de configuração
    if (identifier == 0) {
        throw new FurbRaConfigException(
                      FurbRaConfigException.CONFIGURATION_FILE_NOT_FOUND,
                      msg);
    }

    try {
        XmlResourceParser xml = context.getResources().getXml(identifier);
        values = readXML(xml);
    } catch (XmlPullParserException | IOException e) {
        throw new FurbRaConfigException(
                    FurbRaConfigException.PROBLEM_PARSING_FILE, msg, e);
    }
}
```

### 3.3.3 A integração com a biblioteca de RA

A classe `VuforiaUpdateCallback` é o motor do núcleo de RA da biblioteca FurbRA-library já que é através desta que ocorre a inicialização e gerenciamento do ciclo de execução da biblioteca Vuforia. Esta classe tem sua estrutura fortemente baseada na classe `SampleApplicationSession` existente no aplicativo Android VuforiaSamples 4-0-103 oferecida pela QualComm como fonte de exemplos aos desenvolvedores.

A execução da classe `VuforiaUpdateCallback` inicia pelo método `initAr`, que tem como função definir e armazenar as principais informações que serão utilizadas no decorrer

da aplicação no que se trata da RA, conforme pode ser observado na implementação da rotina no Quadro 11.

Quadro 11 - Implementação do método `initAR`

```
public void initAR(FurbRaBaseActivity activity) {
    VuforiaException = null;
    this.activity = activity;

    //Armazenamento da orientação do dispositivo
    updateActivityOrientation();

    //Armazenamento das dimensões da tela do dispositivo
    storeScreenDimensions();

    //Inicialização assincrona da biblioteca
    if (initVuforiaTask != null) {
        vuforiaException = new VuforiaException(
                                VUFORIA_ALREADY_INITIALIZATED, logMessage);
    }else{
        try {
            (initVuforiaTask = new InitVuforiaTask()).execute();
        } catch (Exception e) {
            vuforiaException = new VuforiaException(
                                INITIALIZATION_FAILURE, logMessage);
        }
    }

    if (vuforiaException != null)
        this.activity.onInitARDone(vuforiaException);
}
```

Primeiramente, a referência à activity `FurbRaBaseActivity` é armazenada na instância. Em seguida os dados do dispositivo como orientação e tamanho da tela são armazenados. Tais dados serão utilizados posteriormente na definição das configurações de renderização da imagem da câmera. Em seguida é feita uma verificação simples para se assegurar que a biblioteca Vuforia não havia sido iniciada previamente, sem ter sido finalizada adequadamente. No caso desta verificação ser positiva a rotina cria uma exceção que representa o problema. Caso contrário o sistema inicializa a *inner class* `InitVuforiaTask`, que possui como finalidade o carregamento da biblioteca em paralelo a `thread` principal, impedindo desta forma o bloqueio da mesma. Havendo qualquer problema com a inicialização do processo assíncrono o sistema cria uma exceção que representa a situação. Ao final da execução do método é verificado se alguma exceção foi criada durante o mesmo, caso seja positivo o processo de inicialização será abreviado, passando a exceção criada para o método `onInitArDone` da instância `FurbRaBaseActivity`.

A partir da chamada ao método `execute` da inner class `InitVuforiaTask` é iniciado um novo processo em paralelo. Este processo inicia com o método `doInBackground` que representa o corpo de execução do novo processo, como pode ser observado no Quadro 12.

Quadro 12 - Implementação do método `doInBackGround` da *inner class* `InitVuforiaTask`

```
protected Boolean doInBackground(Void... params) {
    // Bloco sincronizado para prevenir sobreposição dos métodos de
    // inicialização e finalização.
    synchronized (shutdownLock) {
        //Informa à biblioteca a Activity que irá rodar a RA, as flags de
        //inicialização da mesma e a licença do aplicativo
        Vuforia.setInitParameters(activity, vuforiaFlags,
               FurbRaConfig.getInstance(activity).getVuforiaLicenseKey());

        do {
            //Vuforia.init() bloqueia a execução até um progresso na
            //execução ter ocorrido retornando valores de 0 a 100, que
            //representam a conclusão da inicialização.
            //Este método retorna -1 ou menor no caso de erros, sendo que
            //este valor negativo representa o motivo da falha.
            //A inicialização acaba quando chegar a 100%.
            progressValue = Vuforia.init();

            // Sai do loop nos seguintes casos:
            //a) O processo foi cancelado
            //b) O processo chegou a 100%
            //c) Ocorreu um erro na inicialização da biblioteca.
        } while (!isCancelled() && progressValue >= 0
                && progressValue < 100);

        return (progressValue > 0);
    }
}
```

O processo ocorre inteiramente dentro de um bloco sincronizado para impedir sobreposição da execução do método a outros trechos do código, como a finalização da biblioteca. Este processo é necessário tendo em vista que muitos dos recursos utilizados no processo de inicialização serão também utilizados no processo contrário. A chamada ao método estático `setInitParameters` da classe `Vuforia` serve para definir parâmetros iniciais necessários à biblioteca, como é o caso da *activity* que irá rodar a biblioteca, os *flags* de inicialização da mesma e a licença do aplicativo desenvolvido fornecida pela QualComm, devendo essa ser informada pelo desenvolvedor no arquivo de configurações da biblioteca `furbra_config.xml`. Caso o desenvolvedor falhe em informar uma licença válida para a aplicação, a biblioteca não será inicializada. A inicialização da biblioteca ocorre dentro de uma iteração que a cada ciclo chama o método estático `init` da classe `Vuforia`. Este método realiza a inicialização da biblioteca, sendo que sua execução bloqueia o processo atual até haver sido realizado algum progresso no carregamento da biblioteca. Nesse momento o método desbloqueia o processo e retorna normalmente, um número inteiro de 0 a 100 que representa o percentual de conclusão do carregamento. Em situações adversas o método pode retornar valores negativos que representam erros nesse processo. A falta de licença, mencionada anteriormente, é um dos erros que são identificados e retornados pelo método

`init` como um inteiro negativo. Após o processo ter sido concluído a *inner class* `InitVuforiaTask` procede ao método `onPostExecute`. Este método tem como objetivo verificar o resultado do processo de inicialização da biblioteca, definindo os processos seguintes com base nessa informação, conforme pode ser observado no Quadro 13.

Quadro 13 - Implementação do método `onPostExecute` da *inner class* `InitVuforiaTask`

```
protected void onPostExecute(Boolean result) {
    VuforiaException vuforiaException = null;

    if (result) {
        //Caso a biblioteca tenha sido inicializada corretamente

        // Chama o método responsável pela configuração inicial dos
        // marcadores na activity
        boolean initTrackersResult = activity.doInitTrackers();

        if (initTrackersResult) {
            //Caso os marcadores tenham sido configurados coretamente
            try {
                //Carregamento assincrono das informações dos marcadores
                loadTrackerTask = new LoadTrackerTask();
                loadTrackerTask.execute();
            } catch (Exception e) {
                vuforiaException = new VuforiaException(
                        VuforiaException.LOADING_TRACKERS_FAILURE,
                        logMessage);
            }
        } else {
            vuforiaException = new VuforiaException(
                    VuforiaException.TRACKERS_INITIALIZATION_FAILURE,
                    logMessage);
        }
    } else {
        //Caso a biblioteca não tenha sido inicializada corretamente

        // Obtem-se o motivo especifico da falha no carregamento de
        // maneira a apresentar no log
        String logMessage = getInitializationErrorString(progressValue);

        vuforiaException = new VuforiaException(
                VuforiaException.INITIALIZATION_FAILURE,
                logMessage);
    }

    if (vuforiaException != null) {
        // Finaliza o processo de inicialização com a exceção
        // representando o erro
        activity.onInitARDone(vuforiaException);
    }
}
```

Se o processo de inicialização da biblioteca foi realizado corretamente o primeiro passo da rotina será a chamada ao método `doInitTrackers` da classe `FurbRaBaseActivity`. Este método tem como objetivo o carregamento e configurações das classes da biblioteca Vuforia responsáveis pelos tipos de marcadores que serão detectados. Em seguida, caso esse

processo ocorra sem erros, será utilizado um novo processo assíncrono, desta vez para o carregamento das informações dos marcadores. Caso tenha ocorrido algum erro na inicialização da biblioteca, na configuração das classes dos marcadores ou no início do novo processo assíncrono a rotina irá criar uma exceção sinalizando o problema e o processo será abreviado, passando a exceção criada para o método `onInitArDone` da instância `FurbRaBaseActivity`.

A *inner class* `LoadTrackerTask` é a classe responsável pelo segundo processo assíncrono, no qual serão carregadas as informações dos marcadores. Este processo ocorre paralelamente pelo fato que os dados que definem os marcadores podem estar contidos nos mais diversos formatos que podem necessitar de longos processos de I/O para aquisição. Assim como o processo anterior este processo se inicia no método `doInBackground` (Quadro 14).

Quadro 14 - Implementação do método `doInBackground` da *inner class* `LoadTrackerTask`

```
protected Boolean doInBackground(Void... params) {
    // Bloco sincronizado para previnir sobreposição dos métodos de
    // inicialização e finalização:
    synchronized (shutdownLock) {
        // Carrega os dados dos marcadores.
        return activity.doLoadTrackersData();
    }
}
```

Este processo segue o mesmo modelo do processo anterior, no qual a execução ocorre dentro de um bloco sincronizado a fim de evitar a sobreposição da execução do método a outros trechos do código, como a finalização da biblioteca. A execução do método possui como principal, e única, atribuição a chamada ao método `doLoadTrackersData` da classe `FurbRaBaseActivity`. A responsabilidade por obter os dados dos marcadores e a forma como esses dados serão obtidos, será da classe que estende a `FurbRaBaseActivity`. Este método retorna um booleano indicando se o carregamento dos dados ocorreu sem erros.

A seguir o processo executa o método `onPostExecute` da mesma classe, que possui como objetivo a verificação do processo de carregamento dos dados dos marcadores e o encaminhamento do processo, como pode ser observado no Quadro 15.

Quadro 15 - Implementação do método `onPostExecute` da *inner class* `LoadTrackerTask`

```
protected void onPostExecute(Boolean result) {
    VuforiaException vuforiaException = null;

    if (!result) {
        vuforiaException = new VuforiaException(
                VuforiaException.LOADING_TRACKERS_FAILURE,
                logMessage);
    } else {
       System.gc();

        // Registra com a biblioteca a classe para ser atualizada a cada
        // ciclo através do método QCAR_onUpdate
        Vuforia.registerCallback(VuforiaUpdateCallback.this);

        // Marca a inicilização como concluida.
        started = true;
    }

    // Finaliza o processo de inicialização da biblioteca;
    activity.onInitARDone(vuforiaException);
}
```

Este método verifica o estado de execução do método de carregamento dos dados dos marcadores. No caso de ter sido concluída o método cria uma exceção de modo a notificar esse problema. Caso o processo tenha sido concluído sem erros, a rotina solicita ao *garbage collector* a sua execução. Este processo é realizado pelo fato de que, se a obtenção dos dados dos marcadores ocorreu através de I/O, através de arquivos por exemplo, é possível que resultante desse processo exista uma quantidade significativa de objetos em memória aptos a serem eliminados. A seguir a `outer class VuforiaUpdateCallback` é registrada com a biblioteca Vuforia, de modo que a classe passe a receber notificações, no método `QCAR_onUpdate` do estado da RA gerenciado pela biblioteca. A seguir a *flag started* é alterada para `true` de modo a representar o término do processo de inicialização da biblioteca. Em seguida o método `onInitARDone` da classe `FurbRaBaseActivity` é executado de modo que as configurações necessárias após a inicialização da biblioteca sejam realizadas.

Ainda dentro da classe `VuforiaUpdateCallback`, o método `startAr` tem como função a preparação e configuração da câmera do dispositivo, conforme pode ser observado no Quadro 16.

Quadro 16 - Implementação do método startAR da classe VuforiaUpdateCallback

```
public void startAR() throws VuforiaException {
    String error;

    // Verifica já havia sido realizado a câmera ja esta iniciada
    if (cameraRunning) {
        throw new VuforiaException(
                VuforiaException.CAMERA_INITIALIZATION_FAILURE, errorMsg);
    }

    //Obtem e inicializa o recurso
    if (!CameraDevice.getInstance().init(cameraDevice)) {
        throw new VuforiaException(
                VuforiaException.CAMERA_INITIALIZATION_FAILURE, errorMsg);
    }

    // Configura a imagem de fundo com os dados de tamanho de tela
    // armazenados no método initAR
    configureVideoBackground();

    //Define o modo de vídeo do dispositivo
    if (!CameraDevice.getInstance().selectVideoMode(
            CameraDevice.MODE.MODE_DEFAULT)) {
        throw new VuforiaException(
                VuforiaException.CAMERA_INITIALIZATION_FAILURE, errorMsg);
    }

    //Inicia o equipamento
    if (!CameraDevice.getInstance().start()) {
        throw new VuforiaException(
                VuforiaException.CAMERA_INITIALIZATION_FAILURE, errorMsg);
    }

    //Inicia os marcadores na Activity
    activity.doStartTrackers();

    // Atualiza  a flag que indica o status da câmera
    cameraRunning = true;

    //Tenta configurar o foco da câmera como automático
    if (!CameraDevice.getInstance().setFocusMode(
                        CameraDevice.FOCUS_MODE.FOCUS_MODE_TRIGGERAUTO)) {

        //Em casos em que não existe o modo automático de foco, o
        // equipamento é configurado como foco padrão do equipamento
        CameraDevice.getInstance().setFocusMode(
                        CameraDevice.FOCUS_MODE.FOCUS_MODE_NORMAL);

    }
}
```

Para que a biblioteca FurbRA-library gerencie os recursos durante o ciclo de vida da *activity*, conforme as recomendações de desenvolvimento da plataforma Android (GOOGLE, 2015b), é necessário que tais recursos, como é o caso da câmera, sejam liberados durante as pausas ou na finalização da aplicação. Desta forma, como a classe `VuforiaUpdateCallback` é responsável pelos recursos intrínsecos a RA, a mesma classe também deve estar preparada

para tais mudanças de estado de execução. Com esse propósito foram criados os métodos `pauseAR` e `stopAR`. Tais métodos são responsáveis pela liberação desses recursos além de notificarem a biblioteca Vuforia sobre a mudança na execução da aplicação, conforme pode ser observado nos Quadros 17 e 18.

Quadro 17 - Implementação do método `pauseAR` da classe `VuforiaUpdateCallback`

```
public void pauseAR() {
    if (started) {
        stopCamera();
    }

    Vuforia.onPause();
}
```

Quadro 18 - Implementação do método `stopAR` da classe `VuforiaUpdateCallback`

```
public void stopAR() throws VuforiaException {
    //Finaliza asyncTasks de inicialização que podem estar em execução.
    if (initVuforiaTask != null &&
        initVuforiaTask.getStatus() != InitVuforiaTask.Status.FINISHED){
        initVuforiaTask.cancel(true);
        initVuforiaTask = null;
    }
    if (loadTrackerTask != null &&
        loadTrackerTask.getStatus() != LoadTrackerTask.Status.FINISHED) {
        loadTrackerTask.cancel(true);
        loadTrackerTask = null;
    }

    initVuforiaTask = null;
    loadTrackerTask = null;

    started = false;

    //Libera o recurso de câmera
    stopCamera();

    /*
     * Obtem lock do objeto de sincronia de modo a executar a finalizaçao
     * do SDK sincronamente de maneira  a assegurar que a rotina não se
     * sobrepõe ao carregamento dos marcadores ou a inicialização da
     * biblioteca.
     */
    synchronized (shutdownLock) {

        boolean unloadTrackersResult;
        boolean deinitTrackersResult;

        //Destrói os dados de marcadores já carregados.
        unloadTrackersResult = activity.doUnloadTrackersData();

        //Finaliza os gerenciadores de marcadores
        deinitTrackersResult = activity.doDeinitTrackers();

        // Finaliza o SDK Vuforia
        Vuforia.deinit();

        if (!unloadTrackersResult)
            throw new VuforiaException(
                    VuforiaException.UNLOADING_TRACKERS_FAILURE,
                    errorMsg);

        if (!deinitTrackersResult)
            throw new VuforiaException(
                    VuforiaException.TRACKERS_DEINITIALIZATION_FAILURE,
                    errorMsg);
    }
}
```

Com propósito similar, o método `resumeAR` permite reobter os recursos liberados quando a aplicação continuar a execução. A verificação com a *flag* `started` é realizado já que o método de `resumeAR` (Quadro 19) é chamado pelo método `onResume` e tal método (seção 2.3.4) é realizado não somente a cada retorno de execução (após uma pausa na aplicação) como também no início da *activity*, após o método `onCreate`.

Quadro 19 - Implementação do método `resumeAR` da classe `VuforiaUpdateCallback`

```
public void resumeAR() throws VuforiaException {
    Vuforia.onResume();
    if (started) {
        startAR();
    }
}
```

### 3.3.4 `Activity` base

Como toda lógica de aplicação de um aplicativo Android roda sobre uma *activity*, foi desenvolvida a classe abstrata `FurbRaBaseActivity` que visa a integração com a biblioteca Vuforia através da classe `VuforiaUpdateCallback` e define a estrutura básica para a execução da RA no aplicativo Android. A execução da classe `FurbRaBaseActivity`, inicia pela execução do método `onCreate` (Quadro 20), que possui como principal atribuição a criação do objeto `VuforiaUpdateCallback`.

Quadro 20 - Implementação do método `onCreate` da classe `FurbRaBaseActivity`

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    //Criação da camada superior a imagem da câmera
    addOverlayView();

    // Configuração necessária para manter tela do dispositivo ligada
    // durante a activity
    getWindow().setFlags(
            WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON,
            WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

    this.vuforiaUpdateCallback = new VuforiaUpdateCallback();
}
```

A seguir é executado o método `onResume` (Quadro 21), que tem por principal objetivo inicializar a classe `VuforiaUpdateCallback` e a `GLSurfaceView` (seção 2.3.4), sendo esta referenciada pela variável `glView`. Na primeira execução, após o método `onCreate`, a variável `glView` ainda não terá valor, portanto é necessário realizar a verificação quanto a referência nula. O método também tem como função a criação e posicionamento da camada superior, sendo esta um objeto `RelativeLayout`, que possibilita a inserção de componentes nativos a plataforma Android, como botões, campos de texto, etc, sobre a camada de renderização além de permitir gerenciar os eventos de toques em tela.

Quadro 21 - Implementação do método `onResume` da classe `FurbRaBaseActivity`

```
protected void onResume() {
    super.onResume();

    if (!this.vuforiaUpdateCallback.isARRunning()) {
        this.vuforiaUpdateCallback.initAR(this);
    }

    vuforiaUpdateCallback.resumeAR();

    // Reinicia a view
    if (glView != null) {
        glView.onResume();
    }
}
```

O método `onInitARDone` (Quadro 22), é executado durante o processo de inicialização da biblioteca e é responsável pela inicialização da `view`.

Quadro 22 - Implementação do método `onInitARDone` da classe `FurbRaBaseActivity`

```
public void onInitARDone(VuforiaException exception) {
    if (exception == null) {
        //Cria a view OpenGL
        glView = new FurbRaGLView(this);

        //Adiciona a view na Activity
        addContentView(glView, new ViewGroup.LayoutParams(
                ViewGroup.LayoutParams.MATCH_PARENT,
                ViewGroup.LayoutParams.MATCH_PARENT));

        //Atualiza a camada superior para transparente e a tras para cima
        prepareOverlayView();

        vuforiaUpdateCallback.startAR();
    } else {
        throw exception;
    }
}
```

Os eventos de alterações no ciclo de vida da *activity* `onPause` (Quadro 23) e `onDestroy` (Quadro 24) são sobrescritos na classe `FurbRaBaseActivity`, e tem como principal atribuição a notificação da classe `VuforiaUpdateCallback`, e indiretamente a biblioteca Vuforia, quanto a essas alterações. Na implementação da classe `FurbRaBaseActivity` o método `onDestroy` foi selecionado para notificar o fim da necessidade da RA, em relação ao método `onStop`, tendo em vista que o processo de reinicialização da biblioteca seria muito dispendioso de ocorrer após cada ação do usuário fora do aplicativo. Deste modo os dados carregados pela biblioteca somente serão eliminados na finalização do aplicativo ou da *activity*.

Quadro 23 - Implementação do método `onPause` da classe `FurbRaBaseActivity`

```
protected void onPause() {
    super.onPause();

    if (glView != null) {
        glView.onPause();
    }

    vuforiaUpdateCallback.pauseAR();
}
```

Quadro 24 - Implementação do método `onDestroy` da classe `FurbRaBaseActivity`

```
protected void onDestroy() {
    super.onDestroy();

    vuforiaUpdateCallback.stopAR();
}
```

A classe `FurbRaBaseActivity` também oferece métodos facilitadores para funções normalmente necessárias na criação de um *activity* voltada a RA. Entre esses métodos pode ser citado o método `showToast` (Quadro 25) que tem por objetivo a exibição de mensagens simples em tela e o método `getDisplayMetrics` (Quadro 26) que tem por objetivo obter os dados em relação à tela do dispositivo, como tamanho, densidade de pixels, etc.

Quadro 25 - Implementação do método `showToast` da classe `FurbRaBaseActivity`

```
protected void showToast(String text) {
    Toast.makeText(this, text, Toast.LENGTH_SHORT).show();
}
```

Quadro 26 - Implementação do método `getDisplayMetrics` da classe `FurbRaBaseActivity`

```
protected DisplayMetrics getDisplayMetrics() {
    DisplayMetrics metrics = new DisplayMetrics();
    getWindowManager().getDefaultDisplay().getMetrics(metrics);
    return metrics;
}
```

A classe também define uma quantidade de métodos abstratos, cujo objetivo é definir o comportamento das suas subclasses. Esses métodos podem ser agrupados em 3 conjuntos:

a) gerenciamento de marcadores (Quadro 27): métodos que possuem o objetivo de gerir os dados dos marcadores e os gerenciadores de marcadores fornecidos pela biblioteca Vuforia. Integram esse grupo os seguintes métodos:
   - `doInitTrackers`: tem por objetivo carregar os gerenciadores de marcadores da biblioteca Vuforia,
   - `doLoadTrackersData`: tem por objetivo o carregamento dos dados dos marcadores,
   - `doStartTrackers`: tem por objetivo iniciar os gerenciadores de marcadores carregados no método `doInitTrackers`,
   - `doStopTrackers`: tem por objetivo parar os gerenciadores de marcadores iniciados no método `doStartTrackers`,

- `doUnloadTrackersData`: tem por objetivo limpar os dados dos marcadores carregados no método `doLoadTrackersData`,
- `doDeinitTrackers`: tem por objetivo finalizar os gerenciadores de marcadores carregados no método `doInitTrackers`;

Quadro 27 - Métodos abstratos que definem o gerenciamento de marcadores nas subclasses de `FurbRaBaseActivity`

```
public abstract boolean doInitTrackers();

public abstract boolean doLoadTrackersData();

public abstract boolean doStartTrackers();

public abstract boolean doStopTrackers();

public abstract boolean doUnloadTrackersData();

public abstract boolean doDeinitTrackers();
```

b) Notificações quanto a RA (Quadro 28): métodos que possuem como objetivo notificar quanto a atual situação da RA. Integram esse grupo os seguintes métodos:
   - `trackerFound`: tem por objetivo notificar a aplicação quanto a marcadores identificados na imagem da câmera. O método recebe como parâmetros um objeto `TrackableResult`, classe nativa da biblioteca Vuforia que representa o marcador encontrado, e uma matriz de transformação baseada no posicionamento tridimensional do marcador que pode ser utilizada para renderização,
   - `onARUpdate`: método notificado a cada ciclo pela biblioteca Vuforia. Recebe como parâmetro um objeto `State`, classe nativa da biblioteca Vuforia que tem por objetivo servir como representação à situação da RA gerenciada pela biblioteca Vuforia. Este objeto mantém a informação quanto aos marcadores registrados que estão sendo buscados e os marcadores que foram encontrados no último frame;

Quadro 28 - Métodos abstratos utilizados nas notificações de RA nas subclasses de `FurbRaBaseActivity`

```
public abstract void trackerFound(TrackableResult trackable,
float[] matrix);

public abstract void onARUpdate(State s);
```

c) Renderização (Quadro 29): métodos que possuem como objetivo configurar, ou realizar o processo de renderização. Integram esse grupo os seguintes métodos:
   - `prepareRendering`: tem por objetivo realizar a configuração necessária a renderização como definição do *viewport*, criação da matriz de projeção, etc. Executado no início da *activity* e quando foram realizadas mudanças na

orientação do dispositivo. Como parâmetro o método recebe dois valores inteiros que representam, respectivamente, a largura e a altura da tela,

- `draw`: método responsável por renderizar os objetos gráficos, pela aplicação, através da utilização de chamadas à API OpenGL ES,
- `postDraw`: método executado após o processo de renderização. Tem por finalidade a finalização de recursos utilizados pelo processo de renderização.

Quadro 29 - Métodos abstratos utilizados no processo de renderização nas subclasses de `FurbRaBaseActivity`

```
public abstract void prepareRendering(int width, int height);

public abstract void draw();

public abstract void postDraw();
```

### 3.3.5 *Activity* para detecção de marcadores *Frame*

A classe `FurbRaFrameActivity` é uma classe abstrata, especialização da classe `FurbRaBaseActivity`, orientada a detecção de marcadores do tipo frame.

A classe possui dois métodos destinados a manipulação dos marcadores rastreados. São eles:

a) `addMarker` (Quadro 30): método que tem por finalidade alimentar a lista de marcadores que será utilizado posteriormente, no método `doLoadTrackersData`, para informar a biblioteca os *frames* rastreados. O método recebe por parâmetro um inteiro que representa o identificador numérico do marcador frame, uma `String` que representa o nome do marcador e um objeto `Vec2F`, classe nativa da biblioteca Vuforia que representa um vetor de duas posições do tipo `float`, representando o tamanho do marcador real em unidades de cena;

Quadro 30 - Implementação do método `addMarker` da classe `FurbRaFrameActivity`

```
protected void addMarker(int id, String name, Vec2F size) {
    this.newMarkers.add(new FrameInfo(id, name, size));
    if (this.trackersStarted) {
        doLoadTrackersData();
    }
}
```

b) `removeMarker` (Quadro 31): método que tem por finalidade remover um marcador da lista de marcadores rastreados. Como parâmetro este método recebe um número inteiro que representa o identificador do frame que deve ser removido da lista de marcadores rastreados.

Quadro 31 - Implementação do método `removeMarker` da classe `FurbRaFrameActivity`

```
protected void removeMarker(int id) {
    // Se o marcador já foi informadao ao tracker
    if (dataSet.containsKey(id)) {
        TrackerManager tManager = TrackerManager.getInstance();
        MarkerTracker markerTracker = (MarkerTracker) tManager
                .getTracker(MarkerTracker.getClassType());
        if (markerTracker != null) {
            markerTracker.destroyMarker(dataSet.get(id));
        }
    }else{
        // Se o marcador ainda não foi informado ao tracker, basta
        // remove-lo da classe
        this.newMarkers.remove(new FrameInfo(id, null, null));
    }
}
```

Para permitir que a classe possa detectar os marcadores do tipo *frame*, os métodos `doInitTrackers`, `doLoadTrackersData`, `doStartTrackers`, `doStopTrackers`, `doUnloadTrackersData` e `doDeinitTrackers` foram implementados conforme os quadros Quadro 32, Quadro 33, Quadro 34, Quadro 35, Quadro 36 e Quadro 37.

Quadro 32 – Implementação do método `doInitTrackers` da classe `FurbRaFrameActivity`

```
public boolean doInitTrackers() {
    // Indica que os marcadores foram carregados corretamente.
    boolean result = true;

    // Inicializa o tracker especializado no marcador frame
    TrackerManager trackerManager = TrackerManager.getInstance();
    Tracker trackerBase = trackerManager.initTracker(MarkerTracker
            .getClassType());
    MarkerTracker markerTracker = (MarkerTracker) (trackerBase);

    //
    if (markerTracker == null) {
        result = false;
    }

    return result;
}
```

Quadro 33 - Implementação do método `doLoadTrackersData` da classe `FurbRaFrameActivity`

```
public boolean doLoadTrackersData() {
    //Obter o tracker do marcador frame iniciado
    TrackerManager tManager = TrackerManager.getInstance();
    MarkerTracker markerTracker = (MarkerTracker) tManager
            .getTracker(MarkerTracker.getClassType());
    if (markerTracker == null)
        return false;

    dataSet = new LinkedHashMap<>();

    //Criar os marcadores a partir da lista de novos marcadores
    for (FrameInfo newMarker : newMarkers) {
        Marker frameMarker = markerTracker.createFrameMarker(
             newMarker.getId(), newMarker.getName(), newMarker.getSize());

        if (frameMarker == null) {
            throw new VuforiaException(
                        VuforiaException.TRACKERS_INITIALIZATION_FAILURE,
                        errorMessage);
        }

        this.dataSet.put(frameMarker.getId(), frameMarker);
    }

    //Limpar a lista de marcadores.
    newMarkers.clear();

    return true;
}
```

Quadro 34 - Implementação do método `doStartTrackers` da classe `FurbRaFrameActivity`

```
public boolean doStartTrackers() {
    TrackerManager tManager = TrackerManager.getInstance();
    MarkerTracker markerTracker = (MarkerTracker) tManager
            .getTracker(MarkerTracker.getClassType());
    if (markerTracker == null) {
        this.trackersStarted = false;
    }else{
        markerTracker.start();
        this.trackersStarted = true;
    }
    return trackersStarted;
}
```

Quadro 35 - Implementação do método `doStopTrackers` da classe `FurbRaFrameActivity`

```
public boolean doStopTrackers() {
    boolean result = true;

    TrackerManager tManager = TrackerManager.getInstance();
    MarkerTracker markerTracker = (MarkerTracker) tManager
            .getTracker(MarkerTracker.getClassType());
    if (markerTracker == null) {
        result = false;
    }else{
        markerTracker.stop();
        this.trackersStarted = false;
    }
    return result;
}
```

Quadro 36 - Implementação do método `doUnloadTrackersData` da classe `FurbRaFrameActivity`

```
public boolean doUnloadTrackersData() {
    boolean result = true;

    TrackerManager tManager = TrackerManager.getInstance();
    MarkerTracker markerTracker = (MarkerTracker) tManager
            .getTracker(MarkerTracker.getClassType());
    if (markerTracker != null) {
        for (Marker marker : dataSet.values()) {
            markerTracker.destroyMarker(marker);
        }
    }
    return result;
}
```

Quadro 37 - Implementação do método `doDeinitTrackers` da classe `FurbRaFrameActivity`

```
public boolean doDeinitTrackers() {
    boolean result = true;

    TrackerManager tManager = TrackerManager.getInstance();
    tManager.deinitTracker(MarkerTracker.getClassType());

    return result;
}
```

### 3.3.6 `Activity` para detecção de textos

A classe `FurbRaTextRecognitionActivity` tem por objetivo a detecção de palavras obtidas através da imagem da câmera. A API de detecção de textos fornecida pela biblioteca Vuforia obriga a configuração das palavras, previamente à execução do aplicativo, através de um arquivo fornecido pela Qualcomm com extensão vwl. A API permite que palavras não existentes nesse arquivo sejam detectadas através de uma lista adicional, podendo conter até 10 mil palavras, fornecida à biblioteca através de um arquivo de texto, com codificação UTF-8, de extensão `lst`.

Os métodos `doInitTrackers` (Quadro 38), `doStartTrackers` (Quadro 39), `doStopTrackers` (Quadro 40), `doUnloadTrackersData` (Quadro 41), `doDeinitTrackers` (Quadro 42) são implementados seguindo a mesma lógica vista na classe `FurbRaFrameActivity`, onde são utilizadas classes da biblioteca Vuforia para realizar a inicialização e a finalização dos gerenciadores de marcadores.

Quadro 38 - Implementação do método `doInitTrackers` da classe `FurbRaTextRecognitionActivity`

```
public boolean doInitTrackers() {
    boolean result = true;

    TrackerManager tManager = TrackerManager.getInstance();
    Tracker tracker = tManager.initTracker(TextTracker.getClassType());
    if (tracker == null) {
        result = false;
    }

    return result;
}
```

Quadro 39 - Implementação do método `doStartTrackers` da classe `FurbRaTextRecognitionActivity`

```
public boolean doStartTrackers() {
    boolean result = true;

    Tracker textTracker = TrackerManager.getInstance().getTracker(
            TextTracker.getClassType());
    if (textTracker != null) {
        textTracker.start();
        configureROI();
    }

    return result;
}
```

Quadro 40 - Implementação do método `doStopTrackers` da classe `FurbRaTextRecognitionActivity`

```
public boolean doStopTrackers() {
    boolean result = true;

    Tracker textTracker = TrackerManager.getInstance().getTracker(
            TextTracker.getClassType());
    if (textTracker != null)
        textTracker.stop();

    return result;
}
```

Quadro 41 - Implementação do método `doUnloadTrackersData` da classe `FurbRaTextRecognitionActivity`

```
public boolean doUnloadTrackersData() {
    boolean result = true;

    WordList wl = getWordList();
    if (wl != null) {
        wl.unloadAllLists();
    }
    return result;
}
```

Quadro 42 - Implementação do método `doDeinitTrackers` da classe `FurbRaTextRecognitionActivity`

```
public boolean doDeinitTrackers() {
    boolean result = true;

    TrackerManager tManager = TrackerManager.getInstance();
    tManager.deinitTracker(TextTracker.getClassType());

    return result;
}
```

Destaque-se nesse processo o método `doLoadTrackersData` (Quadro 43). O método inicia pela obtenção do caminho do arquivo `vwl` que deve ser informado pelo desenvolvedor no arquivo de configuração `furbra-config.xml`. Caso o desenvolvedor não tenha informado a localização do arquivo o método retorna informando a falha no carregamento de dados. Caso o desenvolvedor tenha informado no arquivo de configurações o arquivo `lst`, com palavras adicionais este também é fornecido à biblioteca. Caso o desenvolvedor deseje rastrear somente palavras do arquivo `lst`, desconsiderando o vocabulário existente no arquivo `vwl`, o método utiliza as rotinas de filtros da biblioteca Vuforia. A biblioteca Vuforia permite dois tipos de filtros baseados em listas de palavras, lista branca e lista negra, sendo que na primeira o sistema desconsidera todas as palavras não existentes na lista e no segundo são desconsideradas todas as palavras existentes nessa lista. Utilizando o filtro baseado em lista branca a biblioteca consegue fornecer ao desenvolvedor a possibilidade de detectar somente as palavras existentes na lista do arquivo `lst`, sendo que esta implementação está associada à liberação da *flag* `onlyWordsInList` existente na classe.

Quadro 43 - Implementação do método `doLoadTrackersData` da classe `FurbRaTextRecognitionActivity`

```
public boolean doLoadTrackersData() {
    boolean loaded = true;

    String vuforiaDictionaryPath = getVuforiaDictionaryPath();

    WordList wl = getWordList();
    if (wl != null && vuforiaDictionaryPath != null &&
         !vuforiaDictionaryPath.isEmpty()) {
        //Carrega as palavras do dicionário vwl
        loaded = wl.loadWordList(vuforiaDictionaryPath,
                STORAGE_TYPE.STORAGE_APPRESOURCE);

        if ((getWordListPath() != null && !getWordListPath().isEmpty())) {
            //Carrega as palavras adicionais do arquivo lst
            wl.addWordsFromFile(getWordListPath(),
                    STORAGE_TYPE.STORAGE_APPRESOURCE);

            if (isOnlyWordsInList()) {
                // Caso deseja-se desconsiderar as palavras do dicionário
                // e rastrear somente as palavars existentes no arquivo
                // adicional lst
                wl.setFilterMode(
                         WordList.FILTER_MODE.FILTER_MODE_WHITE_LIST);
                wl.loadFilterList(getWordListPath(),
                         STORAGE_TYPE.STORAGE_APPRESOURCE);
            }
        }
    }else{
        loaded = false;
    }

    return loaded;
}
```

A detecção de textos através de imagens pode ser um processo com grande exigência de processamento se esta detecção ocorrer em imagens grandes e/ou com alta resolução. Desta forma a classe fornece métodos que permitem restringir a área de detecção. A classe define o método abstrato `getRoi` que retorna um objeto `Dimension` contendo a posição e o tamanho da área desejada de busca, baseado na tela no dispositivo.

### 3.3.7 Renderização

A camada de renderização da biblioteca FurbRA-library é atribuída a duas classes: `FurbRaGLView` e `FurbRaRenderer`.

A classe `FurbRaGLView`, uma extensão da classe `GLSurfaceView`, tem dois objetivos básicos: a configuração básica necessária à renderização e à instanciação da classe `FurbRaRenderer`, como pode ser observado no Quadro 44.

Quadro 44 - Implementação do método construtor da classe `FurbRaGLView`

```
Public FurbRaGLView(FurbRaBaseActivity context) {
    super(context);

    // Determina a versão do OpenGL ES
    setEGLContextClientVersion(2);

    //Solicita a utilização de formato que permite transparencia
    this.getHolder().setFormat(PixelFormat.TRANSLUCENT);

    renderer = new FurbRaRenderer(context);
    setRenderer(renderer);
    renderer.setActive(true);
}
```

A classe `FurbRaRenderer`, extensão da classe `GLSurfaceView.Renderer` possui uma integração maior com a biblioteca Vuforia tendo em vista que esta utiliza método nativos da mesma. A classe possui três atribuições básicas:

a) notificar a biblioteca Vuforia dos eventos do ciclo de vida da classe de renderização;
b) propagar a execução de métodos de renderização à *activity* principal (baseada na classe `FurbRaBaseActivity`);
c) obter com a biblioteca o estado atual da detecção de marcadores e repassar essa informação à *activity* principal (baseada na classe `FurbRaBaseActivity`).

O método `onSurfaceCreated` (Quadro 45) é um exemplo da primeira atribuição, tendo em vista que sua função principal, e única, é repassar a notificação quanto a criação da classe de renderização à biblioteca Vuforia.

Quadro 45 - Implementação do método `onSurfaceCreated` da classe `FurbRaRenderer`

```
public void onSurfaceCreated(GL10 gl, EGLConfig config) {
     Vuforia.onSurfaceCreated();
}
```

O método `onSurfaceChanged` (Quadro 46), por sua vez, tem dupla atribuição: notificar a biblioteca quanto a mudanças na superfície disponível à renderização e propagar este evento à *activity*, neste caso através do método `prepareRendering`.

Quadro 46 - Implementação do método `onSurfaceChanged` da classe `FurbRaRenderer`

```
public void onSurfaceChanged(GL10 gl, int width, int height) {
    Vuforia.onSurfaceChanged(width, height);

    activity.prepareRendering(width, height);
}
```

O método `onDrawFrame` possui atribuição única de verificar a situação quanto ao estado de ativação do *render* e finalizar a execução da renderização do *frame* caso a *flag*

active esteja desabilitada. Caso a mesma esteja habilitada a execução da renderização é repassada ao método renderFrame.

Quadro 47 - Implementação do método onDrawFrame da classe FurbRaRenderer

```
public void onDrawFrame(GL10 gl) {
    if (!active)
        return;

    renderFrame();
}
```

A execução do método renderFrame possui todas as três atribuições básicas. O método inicia a execução da renderização do *frame* através da chamada à API OpenGL ES glClear que limpa os *buffers*. Em seguida o método notifica o início da renderização à biblioteca Vuforia e solicita à essa a renderização da imagem da câmera no *background* da view. A seguir a renderização é passada para a *activity* através do método draw da classe FurbRaBaseActivity. A seguir os marcadores detectados são obtidos com a biblioteca Vuforia e repassados, individualmente, junto com a respectiva matriz *model-view* criada a partir do marcador detectado, à *activity* através do método trackerFound. O evento postDraw da *activity* e a notificação quanto ao fim da renderização à biblioteca Vuforia são as duas últimas instruções executadas pelo método.

Quadro 48 - Implementação do método `renderFrame` da classe `FurbRaRenderer`

```
private void renderFrame() {
    //Limpa os buffers
    GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT |
                                     GLES20.GL_DEPTH_BUFFER_BIT);

    //Marca o inicio da renderização
    State state = Renderer.getInstance().begin();

    // Solicitar a biblioteca a renderização da imagem da câmera no
    // background
    Renderer.getInstance().drawVideoBackground();

    //Repassar a renderização à Activity
    activity.draw();

    for (int id = 0; id < state.getNumTrackableResults(); id++){
        //Obtem o marcador detectado
        TrackableResult trackableResult = state.getTrackableResult(id);

        //Obtem a matriz modelView baseado no marcador
        Matrix44F modelViewMatrix_Vuforia = Tool
                .convertPose2GLMatrix(trackableResult.getPose());
        float[] modelViewMatrix = modelViewMatrix_Vuforia.getData();

        //Repassa o marcador e a matriz à Activity
        activity.trackerFound(trackableResult, modelViewMatrix);
    }

    //Evento de finalização da renderização
    activity.postDraw();

    //Marca o fim da renderização
    Renderer.getInstance().end();
}
```

### 3.3.8 Mídia

A camada de Mídia da biblioteca `FurbRA-library` é representada pela classe `SoundBox` que permite ao desenvolvedor o carregamento e controle de execução de arquivos de áudio. Esta classe tem sua execução baseada na classe nativa `SoundPool` e é estruturada de forma a permitir o carregamento assíncrono dos arquivos. A classe também permite ao desenvolvedor informar um identificador próprio ao recurso e gerenciar toda a execução do áudio através deste identificador, ao invés do identificador oferecido pela classe `SoundPool`.

O conceito subjacente é que os recursos sonoros necessários à execução de um jogo sejam informados à biblioteca, através do método `addSound`, no início da aplicação ou da *activity*. Este método adiciona todos os arquivos em um *cache* que será processado pelo *framework* quando a aplicação utilizar o método `load`. A partir da execução deste método o carregamento destes recursos será executado pela *inner class* `LoadSoundsTask` (Quadro 49). Esta classe verifica se o recurso já não foi carregado anteriormente, caso negativo irá realizar

o carregamento do mesmo. Caso positivo a classe atualiza a referência para o identificador já carregado. Este processo evita que o mesmo arquivo seja carregado mais de uma vez, consumindo memória desnecessariamente.

Quadro 49 - *Inner class* `LoadSoundTask`

```
private class LoadSoundsTask extends AsyncTask<Context, Void, Void> {
    protected Void doInBackground(Context... params) {
        Map<Integer, Integer> loadedValues = new HashMap();
        //Mapa de ids de resources e ids de pool
        for (Integer resId : new HashSet<>(soundBoard.values())) {
            if (loadedValues.containsKey(resId)) {
                continue;
            }
            //Realize o carregamento do recurso
            int loadId = pool.load(params[0], resId, 1);
            loadedValues.put(resId, loadId);
        }

        //Atualiza o id do soundbourd do ID do res pelo ID do SoundPool
        for (Map.Entry<Integer, Integer> entry : soundBoard.entrySet()) {
            entry.setValue(loadedValues.get(entry.getValue()));
        }

        return null;
    }
}
```

A execução do som será permitida quando o carregamento dos recursos for concluído. Se uma tentativa de utilização dos recursos ocorrer previamente a esse momento, a biblioteca lançara uma exceção do tipo `SoundBoxLoadedException` com o identificador `SOUND_BOX_NOT_LOADED`.

Para informar a aplicação quanto a finalização, e possibilidade de uso dos recursos de áudio, a classe utiliza de uma estrutura de `listener`, na qual a aplicação informa a classe uma instância de objeto que implementa a interface `SoundBoxListener` e o método `resourceLoadingCompleted` dessa interface será executado na finalização do carregamento.

### 3.3.9 Comunicação

A camada de comunicação da biblioteca FurbRA-library é representada pela classe `LocalNetworkCommunication`. Esta classe, desenvolvida segundo o padrão de projeto *singleton,* utiliza da especificação NSD existente na plataforma Android para permitir a publicação e a descoberta das instâncias da aplicação que existem na rede local.

A execução da classe é dividida em três etapas: a publicação do serviço NSD, a obtenção dos serviços na rede local e a troca de mensagens.

A primeira etapa, a publicação do serviço NSD, inicia antes da execução da aplicação, quando o desenvolvedor informa no arquivo de configurações da biblioteca, `furbra-`

config.xml, o nome pelo qual a aplicação será conhecida na rede local e a porta de comunicação que será utilizada para o recebimento de mensagens. Tais informações serão utilizadas na criação do serviço pelo método startService (Quadro 50). Caso alguma dessas informações não seja fornecida, a execução do método resultará no lançamento de uma exception do tipo LocalNetworkCommunicationStartServiceException.

Quadro 50 - Implementação do método startService da classe LocalNetworkCommunication

```
public void startService(Context context) {
    if (this.serviceActive) {
        return;
    }
    this.currentServiceName = "";

    //Obtenção das configurações
    FurbRaConfig furbRaConfig = FurbRaConfig.getInstance(context);
    this.defaultServiceName = furbRaConfig.getNetworkServiceName();
    this.port = furbRaConfig.getNetworkServicePort();

    //Validação
    if (this.defaultServiceName == null ||
        this.defaultServiceName.isEmpty()) {
        throw new LocalNetworkCommunicationStartServiceException(
                       SERVICE_NAME_NOT_INFORMED, msg);
    }else if (this.port == 0) {
        throw new LocalNetworkCommunicationStartServiceException(
                       SERVICE_PORT_NOT_INFORMED,  msg);
    }

    //Criação da informação do serviço
    NsdServiceInfo serviceInfo = prepareServiceInfo();

    //Registro do ip local
    registerLocalAddress(context);

    //Inicia a thread de recebimento de mensagens
    this.serverThread = new Thread(new ServerThread());
    this.serverThread.start();

    //Instancia o listener do registro
    initializeRegistrationListener();

    //Registra o serviço
    this.nsdManager = (NsdManager)
                       context.getSystemService(Context.NSD_SERVICE);
    this.nsdManager.registerService(serviceInfo,
            NsdManager.PROTOCOL_DNS_SD, this.registrationListener);
}
```

O processo de iniciação do serviço NSD ocorre paralelamente ao processo principal, sendo necessário registrar um listener, do tipo NsdManager.RegistrationListener, que será notificado quanto ao progresso do mesmo conforme pode ser observado no Quadro 51.

Quadro 51 - Método `initializeRegistrationListener` da classe `LocalNetworkCommunication`

```
private void initializeRegistrationListener() {
    this.registrationListener = new NsdManager.RegistrationListener() {

        @Override
        public void onServiceRegistered(NsdServiceInfo NsdServiceInfo) {
            /*
             * Embora o nome do serviço seja definido no registro,
             * ele deve ser único na rede, desta forma pode ser alterado
             * pela aplicação no registro.
             */
            currentServiceName = NsdServiceInfo.getServiceName();
            serviceActive = true;

            //Inicialização da descoberta de rede
            startDiscovery();
        }

        @Override
        public void onRegistrationFailed(NsdServiceInfo serviceInfo,
                                             int errorCode) {
            String error = getNSDInicializationErrorMessage(errorCode);
            registrationListener = null;
            serviceActive = false;
        }

        @Override
        public void onServiceUnregistered(NsdServiceInfo arg0) {
            registrationListener = null;
            serviceActive = false;

        }

        @Override
        public void onUnregistrationFailed(NsdServiceInfo serviceInfo,
                                               int errorCode) {
            String error = getNSDInicializationErrorMessage(errorCode);
            // LOG

            registrationListener = null;
            serviceActive = false;
        }
    };
}
```

É interessante ressaltar que cada serviço NSD deve possuir um nome único na rede local. Deste modo é possível que ao registrar o serviço já exista um serviço rodando com o mesmo nome. Nestes casos o serviço será criado acrescentando-se um numeral entre parênteses ao final do nome. Esse processo ocorre automaticamente pela plataforma e somente é verificado pela biblioteca durante a execução do método `onServiceRegistered`, do `listener` de registro, sendo que nesse momento o nome alterado é armazenado.

A segunda etapa, obtenção dos serviços na rede local, inicia assim que ocorre a notificação quanto a inicialização do serviço ao `listener`, via método `startDiscovery`. Embora o processo de descoberta ocorra automaticamente durante o fluxo de registro do

serviço, este processo pode ser gerenciado a qualquer momento da execução do aplicativo pelos métodos `startDiscovery` (Quadro 52) e `stopDiscovery`, sendo necessário somente que o serviço já tenha sido registrado. Tais métodos são fornecidos pois a descoberta de serviços é um processo que demanda muitos recursos pelo aparelho (GOOGLE, 2015c) e é aconselhado sua execução somente durante períodos necessários de maneira a não drenar a bateria do dispositivo.

Quadro 52 - Implementação do método `startDiscovery` da classe `LocalNetworkCommunication`

```
public int startDiscovery() {
    if (!this.serviceActive) {
        return DISCOVERY_FAILURE_SERVICE_NOT_STARTED;
    }
    if (this.discoveryActive) {
        return DISCOVERY_FAILURE_ALREADY_STARTED;
    }

    initializeDiscoveryListener();

    nsdManager.discoverServices(
            SERVICE_TYPE, NsdManager.PROTOCOL_DNS_SD, discoveryListener);

    return DISCOVERY_SOLICITED;
}
```

A execução da descoberta de serviços, assim como o registro do serviço, é um processo que ocorre em paralelo à *thread* principal. Desta forma também é obrigatório informar um *listener*, do tipo `NsdManager.DiscoveryListener`, de maneira a acompanhar o progresso do mesmo, conforme pode ser observado no Quadro 53.

Quadro 53 – Implementação do método `initializeDiscoveryListener` da classe `LocalNetworkCommunication`

```
private void initializeDiscoveryListener() {
    this.discoveryListener = new NsdManager.DiscoveryListener() {

        @Override
        public void onDiscoveryStarted(String regType) {
            discoveryActive = true;
        }

        @Override
        public void onServiceFound(NsdServiceInfo service) {
            String serviceName = fixServiceName(service.getServiceName());
            if (serviceName.equals(currentServiceName)) {
                //Se for o serviço local, nada é feito
            } else if (service.getServiceType().equals(SERVICE_TYPE)
                    && (serviceName.contains(defaultServiceName) ||
                    serviceName.equals(defaultServiceName))) {
                nsdManager.resolveService(service,
                                          new NsdResolveListener());
            }
        }

        @Override
        public void onServiceLost(NsdServiceInfo service) {
            LanService lanService = findServiceByAddress(
                                                  service.getHost());
            removeServiceFromKnowList(lanService);
        }

        @Override
        public void onDiscoveryStopped(String serviceType) {
            discoveryActive = false;
            discoveryListener = null;
        }

        @Override
        public void onStartDiscoveryFailed(String serviceType,
                                           int errorCode) {
            String error = getNSDInicializationErrorMessage(errorCode);
            // LOG

            discoveryActive = false;
            discoveryListener = null;
        }

        @Override
        public void onStopDiscoveryFailed(String serviceType,
                                          int errorCode) {
            String error = getNSDInicializationErrorMessage(errorCode);
            // LOG

            discoveryActive = false;
            discoveryListener = null;
        }
    };
}
```

Assim que um serviço é descoberto, o *listener* verifica se o mesmo trata de um serviço da mesma ou de uma instância diferente da aplicação. Se for o caso de uma outra instância é

executado o processo assíncrono de resolução do serviço onde é obtido o endereço e porta do mesmo. A partir da resolução o serviço é armazenado em uma lista interna de serviços conhecidos. A classe utiliza uma estrutura de *listener*, baseada na interface `LocalExternalServiceDiscoveryListener`, para notificar a aplicação quanto a descoberta e também quanto a perda de comunicação de serviços de rede.

A troca de mensagens na biblioteca é um processo assíncrono, tanto para o envio quanto para o recebimento. O trafego de mensagens é baseado nos dados obtidos a partir da serialização de objetos que herdam da classe abstrata `Mensagem`. O recebimento de mensagens já é possível a partir do registro do serviço, onde o método `startService` inicia o processo paralelo, definido pela *inner class* `ServerThread` (Quadro 54), que tem como único objetivo observar a porta definida na configuração do serviço, quanto ao recebimento de mensagens e repassar essas para a um novo processo paralelo, baseado na *inner class* `IncomeCommunicationThread` (Quadro 55). Esta classe irá serializar a mensagem, obter o remetente da mesma e informar a aplicação quanto ao recebimento. Por se tratar de um processo assíncrono a notificação à aplicação quanto ao recebimento foi desenvolvido na estrutura de *listener,* baseado na interface `LocalNetworkMessageReceivedListener`, de maneira que a aplicação se registra para receber notificações quanto a determinados tipos de mensagens.

Quadro 54 - *Inner class* `ServerThread`

```
private class ServerThread implements Runnable {
    public void run() {
        Socket socket;
        while (!Thread.currentThread().isInterrupted()) {
            try {
                socket = serverSocket.accept();

                // Se não existe listener registrado não existe motivo de
                // continuar o processo.
                if (localNetworkMessageReceivedListeners != null) {
                    IncomeCommunicationThread commThread =
                                    new IncomeCommunicationThread(socket);
                    new Thread(commThread).start();
                }
            } catch (IOException e) {
                e.printStackTrace();
            }
        }
    }
}
```

Quadro 55 – *Inner class* `IncomeCommunicationThread`

```
private class IncomeCommunicationThread implements Runnable {

    private InetAddress inetAddress;
    private ObjectInputStream objectInputStream;

    public IncomeCommunicationThread(Socket socket) {
        try {
            inetAddress = socket.getInetAddress();
            objectInputStream = new ObjectInputStream(
                                                socket.getInputStream());
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    public void run() {
        try {
            Message message = (Message) objectInputStream.readObject();
            objectInputStream.close();
            LanService knowService = findServiceByAddress(inetAddress);
            if (knowService == null) {
                //Se o serviço não é conhecido acrescenta-se esse a lista
                knowService = includeServiceKnowList(
                                        message.getOriginaryLanServiceName(),
                                        inetAddress);
            }
            //Notifica-se os listeners do recebimento
            fireMessageReceivedListeners(knowService, message);
        } catch (ClassNotFoundException | IOException e) {
            e.printStackTrace();
        }
    }
}
```

O envio de mensagens é realizado através do método `sendMessage`. Para esse método é passada a mensagem, uma instância do objeto que herda da classe `Message` e o remetente da mesma, obrigatoriamente um serviço da lista de serviços conhecidos obtido pela aplicação através do método `getKnownServices`. No método de envio, a mensagem e o remetente são repassados a um novo processo paralelo, baseado na *inner class* `OutcomeCommunicationThread` (Quadro 56). Caso no envio da mensagem seja verificado uma indisponibilidade na realização da conexão com o serviço, entende-se que o serviço não está mais disponível e o sistema o remove da lista de serviços conhecidos.

Quadro 56 - I*nner class* `OutcomeCommunicationThread`

```
private class OutcomeCommunicationThread implements Runnable {

    private final LanService lanService;
    private Message message;

    public OutcomeCommunicationThread(LanService lanService,
                                      Message message) throws IOException {
        this.message = message;
        this.lanService = lanService;
    }

    public void run() {
        try {
            Socket socket = new Socket(lanService.getAddress(), port);
            ObjectOutputStream out = new ObjectOutputStream(
                                            socket.getOutputStream());
            out.writeObject(message);
            out.flush();
            out.close();
        } catch (ConnectException e) {
            // Se houve falha de conexão remove-se o serviço das redes
            // conhecidas.
            removeServiceFromKnowList(lanService);
        } catch (IOException e) {
            // LOG
        }
    }
}
```

### 3.3.10 Operacionalidade da implementação

Para demonstrar a operacionalidade da FurbRA-library, foi desenvolvido um jogo de RA denominado Animais. Este jogo tem propósito educativo auxiliando na alfabetização de crianças. Nele, crianças em etapa de alfabetização utilizarão peças com letras para montar o nome de animais que são apresentados na tela do dispositivo. A aplicação também possui as mensagens em tela do jogo mostradas na forma de LIBRAS de maneira a, além de estimular a alfabetização, estimular na criança a vontade do aprendizado de LIBRAS. A aplicação foi desenvolvida com suporte mínimo à plataforma Android 4.1 (Jelly Bean) e utilizando a IDE Android Studio 1.2.1.1.

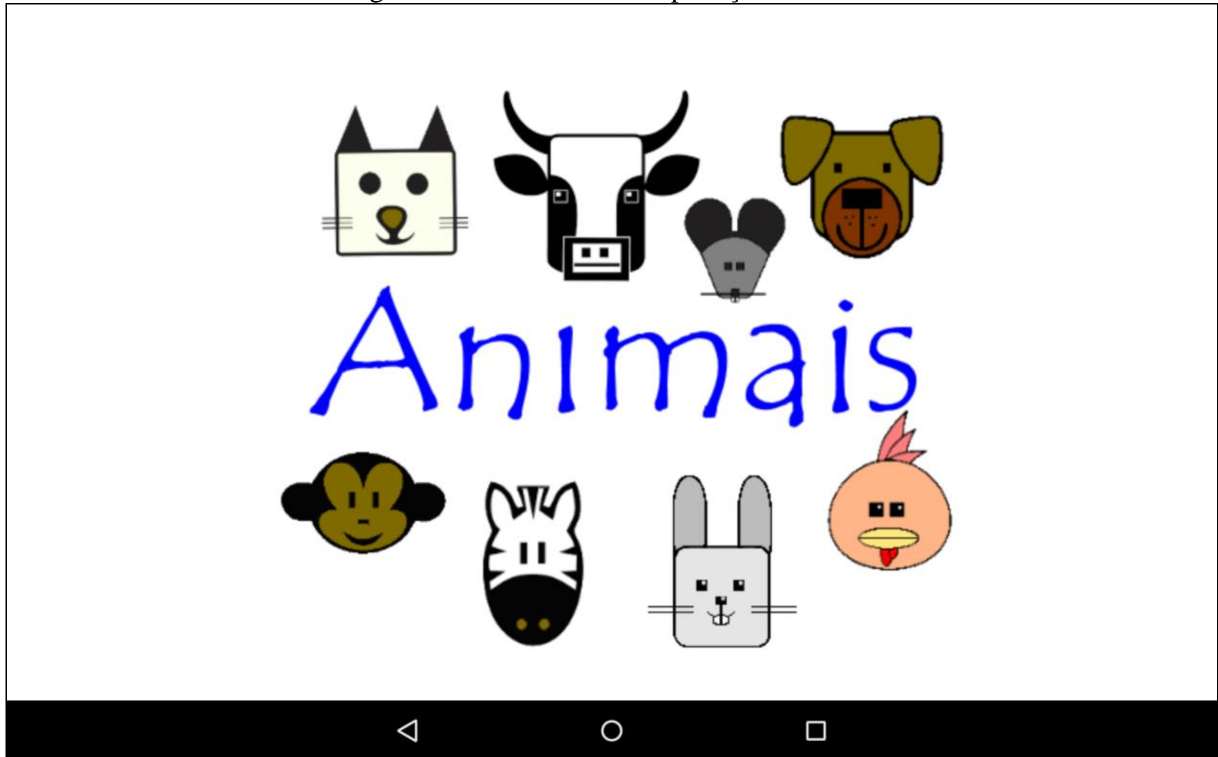
#### 3.3.10.1 Visão geral da aplicação

A aplicação FurbRA-library foi desenvolvida de forma a possuir um menu inicial com opções que permitem iniciar um jogo nos modos *singleplayer* e *multiplayer*. A aplicação possui cinco telas que serão apresentadas a seguir.

3.3.10.1.1 Tela inicial

Quando a aplicação é aberta é apresentada a tela inicial, mostrando o título do aplicativo, conforme é mostrado na Figura 17. Assim que qualquer toque for realizado na tela a próxima tela (menu) é apresentada.

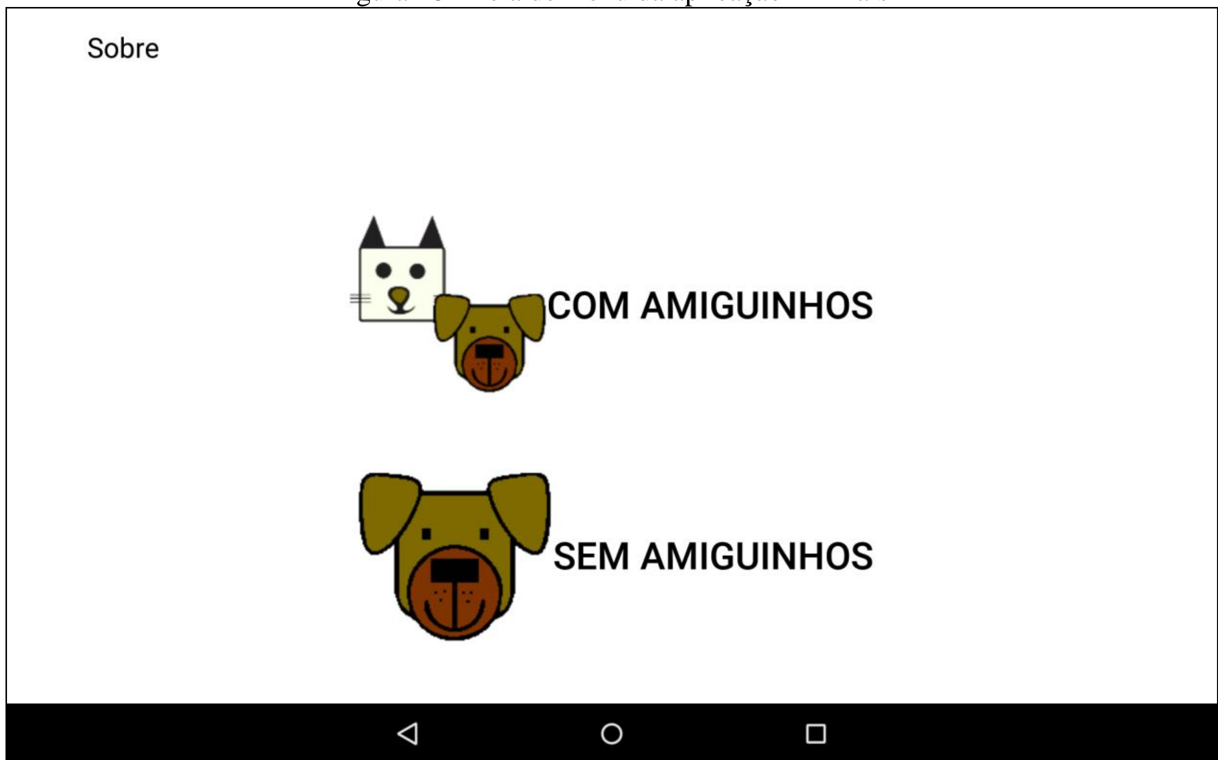
Figura 17 - Tela inicial da aplicação Animais




3.3.10.1.2 Menu

Na tela do menu (Figura 18), são apresentadas as opções da aplicação, sendo elas: `Sobre`, `Com amiguinhos` e `Sem amiguinhos`.

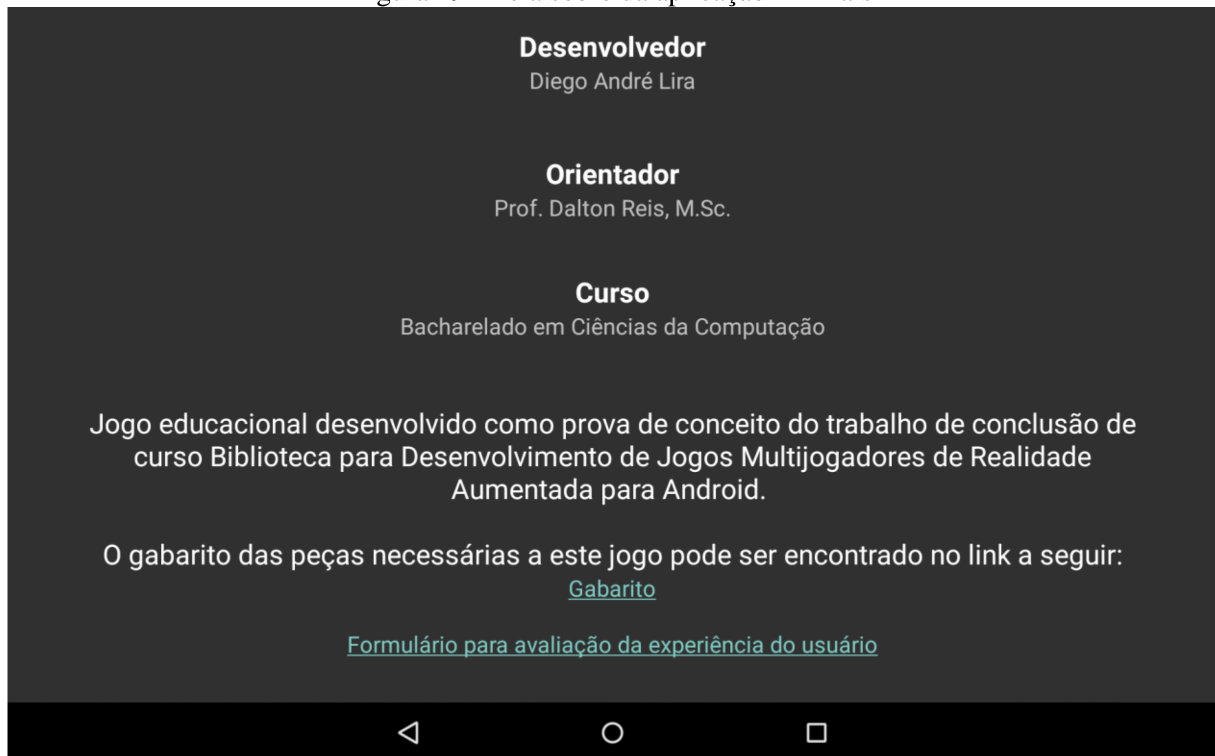
Figura 18 - Tela de menu da aplicação Animais




3.3.10.1.3 `Sobre`

Uma vez selecionada a opção `Sobre` (Figura 19), são apresentados ao usuário os detalhes sobre o desenvolvimento da aplicação tais como a descrição do mesmo, o desenvolvedor, o orientador, etc. Nesta opção também são apresentados dois *links* sendo um deles o gabarito para produção das peças do jogo (Apêndice A) e o outro uma avaliação *online* quanto a experiência do usuário com o aplicativo (Apêndice B).

Figura 19 - Tela sobre da aplicação Animais




#### 3.3.10.1.4 `Sem amiguinhos`

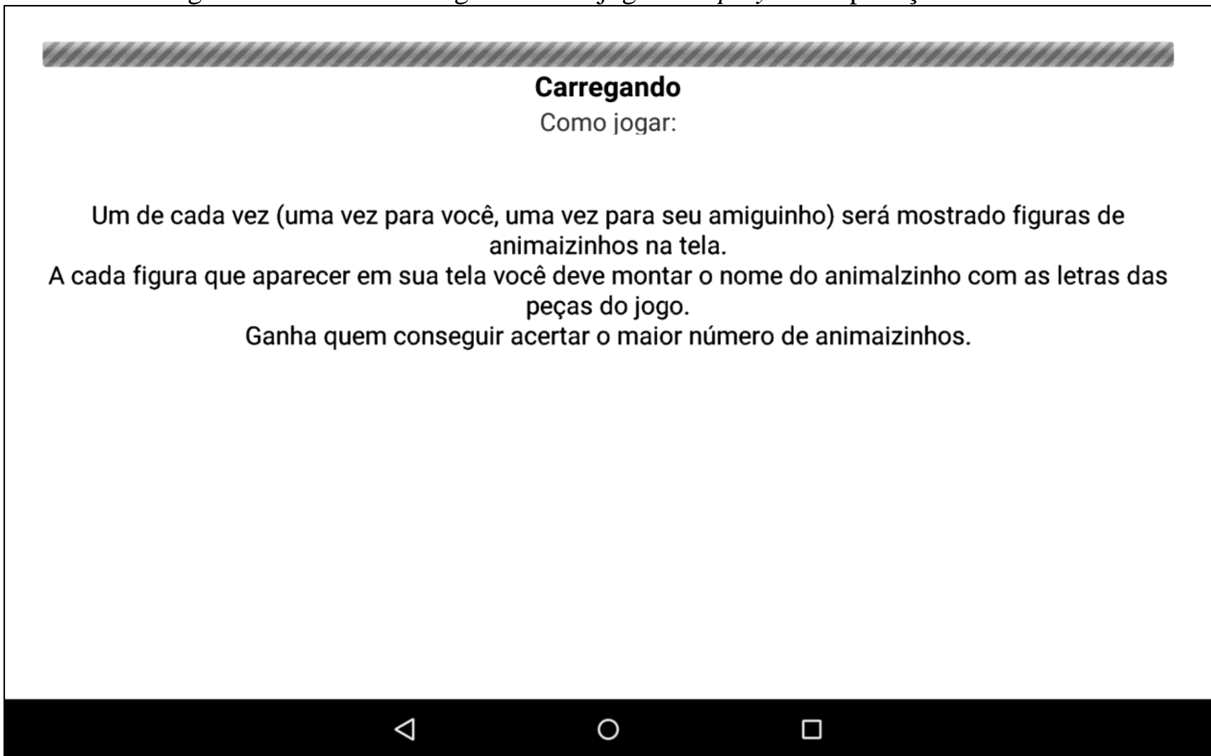
A opção `Sem amiguinhos` inicia um jogo *singleplayer* no qual o usuário deve utilizar as peças, obtidas através do gabarito disponibilizado na opção `Sobre`, para montar o nome de animais que são mostrados em tela. Para facilitar a identificação do usuário para com o animal foi adicionada a possibilidade de escutar o som que o animal faz (ação disparada quando ocorre o toque em tela). O jogo termina quando o usuário consegue identificar todos os oito animais disponibilizados na aplicação.

#### 3.3.10.1.5 `Com amiguinhos`

A opção `Com amiguinhos` inicia um jogo competitivo baseado em turnos. Quando esta opção é pressionada o sistema apresenta uma listagem das instâncias do aplicativo que estão rodando na rede local.

Ao selecionar qualquer um dos usuários é realizado um convite de jogo ao mesmo. Enquanto não houver resposta por parte dos outros usuários é demonstrada uma tela de carregamento. Se o convite for recusado a tela de carregamento é removida de maneira que seja possível realizar um novo convite. Caso haja o aceite o sistema apresenta uma tela de carregamento com as instruções do jogo (Figura 20). Nesse momento a biblioteca de RA é carregada, assim como os recursos gráficos e de áudio.

Figura 20 - Tela de carregamento do jogo *multiplayer* da aplicação Animais



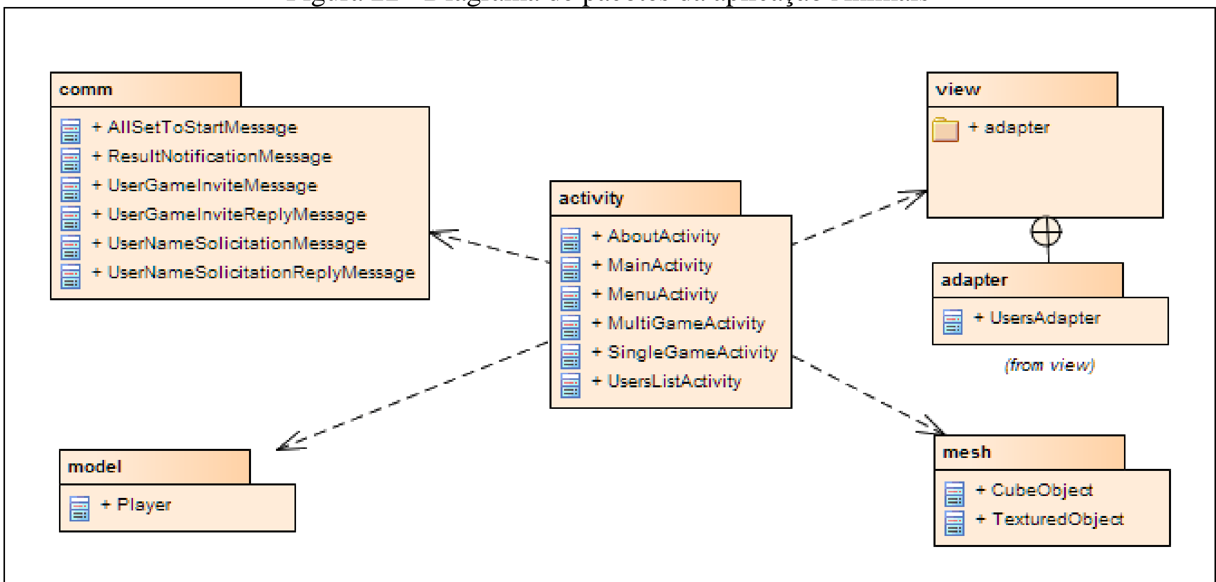

A tela de carregamento será removida e o jogo será iniciado quando os carregamentos em ambos os dispositivos tiverem sido finalizados. Em estrutura de turnos, será apresentado a figura de um animal em um dos dispositivos (Figura 21), enquanto nos demais é apresentada uma mensagem de aguardo. Para auxiliar a verificação do animal por parte do usuário, caso o mesmo toque na tela do dispositivo, o aplicativo emite o som que o animal emite. No dispositivo onde a figura do animal for apresentada o usuário deverá utilizar as peças com as letras de maneira que essas formem o nome do animal mostrado. O usuário deve mostrar a palavra formada para a câmera do dispositivo de modo que a aplicação possa reconhecer se a mesma foi escrita corretamente. O usuário terá dois minutos para executar essa operação corretamente, após esse tempo o usuário perderá o turno. Quando o usuário conseguir acertar a formação da palavra o mesmo será notificado e o turno será passado para o outro usuário. O jogo possui oito animais cadastrados, sendo que um animal nunca é mostrado duas vezes nem para mais de um usuário. O jogo termina quando todos os oito animais forem apresentados aos usuários. Vence aquele que tiver acertado o maior número de animais.

Figura 21 - Aplicação Animais

### 3.3.10.2 Estrutura de classes

Nesta seção são descritas as classes e estruturas desenvolvidas que formam a aplicação Animais. Para facilitar o entendimento de como as classes estão reunidas, na Figura 22 estão sendo demonstrados os pacotes, suas dependências bem como as classes que os compõem.

Figura 22 - Diagrama de pacotes da aplicação Animais



#### 3.3.10.2.1 Pacote `model`

O pacote denominado `model` possui as classes POJO que formam a estrutura de dados. Neste pacote existe a classe `Player` que tem por objetivo representar as informações do jogador adversário no momento de um jogo *multiplayer*.

#### 3.3.10.2.2 Pacote `mesh`

O pacote denominado `mesh` agrupa as classes que tem por função a renderização de imagens. Neste pacote existem as classes `CubeObject`, responsável por renderizar estruturas retangulares de cor sólida, e a classe `TexturedObject`, responsável por renderizar um objeto texturizado.

#### 3.3.10.2.3 Pacote `view`

O pacote denominado `view` agrupa as classes que tem por função ser utilizadas na composição da interface com o usuário. Neste pacote existe a classe `UsersAdapter`, utilizada na composição de tabelas de usuários.

#### 3.3.10.2.4 Pacote `comm`

O pacote denominado `comm` agrupa as classes que tem por função representar mensagens trafegadas entre as aplicações em um jogo *multiplayer*. Estão contidos neste pacote as classes:

a) `AllSetToStartMessage`: Classe que representa uma mensagem notificando a finalização do carregamento dos recursos necessários ao inicio do jogo;

b) `ResultNotificationMessage`: Classe que representa uma mensagem notificando o final da rodada do adversário, informando o resultado da mesma (acerto ou erro);

c) `UserGameInviteReplyMessage`: Classe que representa uma mensagem notificando um convite de jogo por parte de um usuário da rede local;

d) `UserNameSolicitationMessage`: Classe que representa uma mensagem solicitando o nome do usuário atual;

e) `UserNameSolicitationReplyMessage`: Classe que representa uma mensagem notificando a resposta do usuário quanto a um convite de jogo (`UserGameInviteReplyMessage`).

#### 3.3.10.2.5 Pacote `activity`

O pacote denominado `activity` agrupa as classes que representam, e possuem a lógica de negócio, das telas da aplicação. Estão contidos neste pacote as classes que representam as seguintes telas:

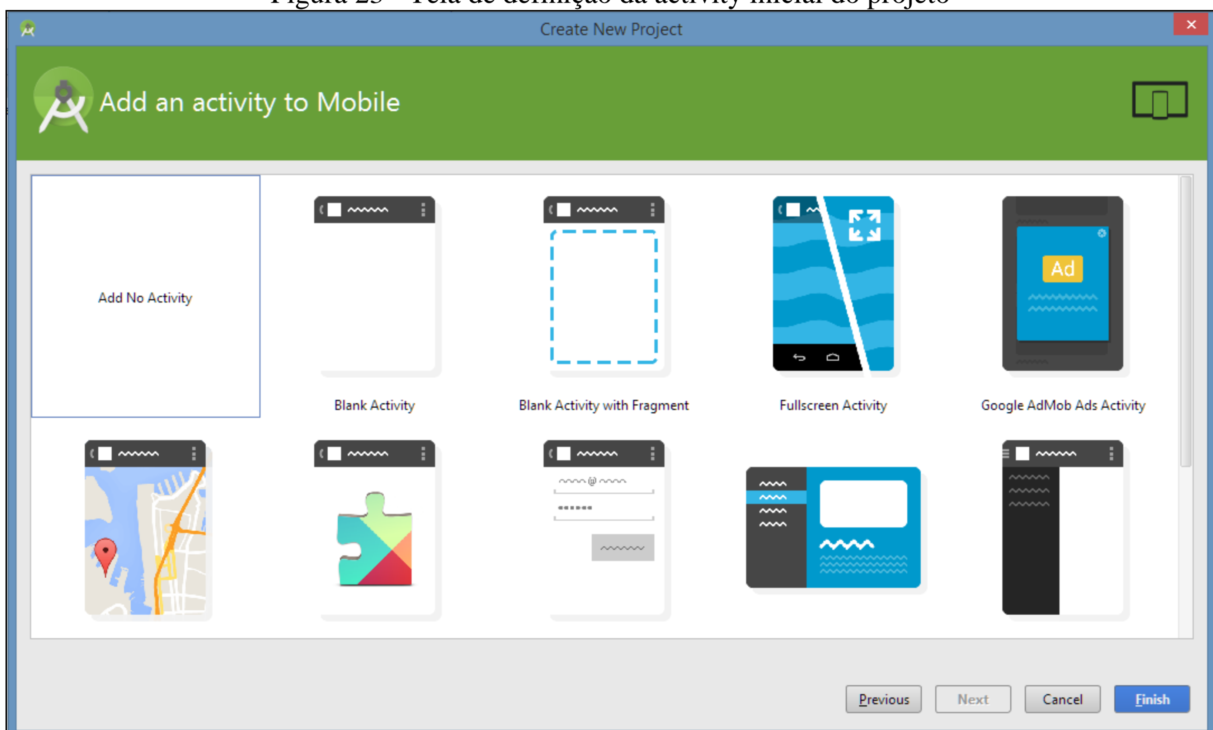
a) `AboutActivity`: Tela `Sobre`;
b) `MainActivity`: Tela inicial da aplicação;
c) `MenuActivity`: Tela de menu inicial;
d) `UserListActivity`: Tela com a listagem de jogadores disponíveis na rede local;
e) `MultiGameActivity`: Tela do jogo *multiplayer*;
f) `SingleGameActivity`: Tela do jogo *singleplayer*.

#### 3.3.10.3 Início do projeto

Antes de se iniciar a implementação o desenvolvedor deve obter uma licença e os arquivos referentes à biblioteca Vuforia. No momento do desenvolvimento deste trabalho ambos podiam ser obtidos, sem custo, através do portal da biblioteca voltado a desenvolvedores (QUALCOMM, 2015b).

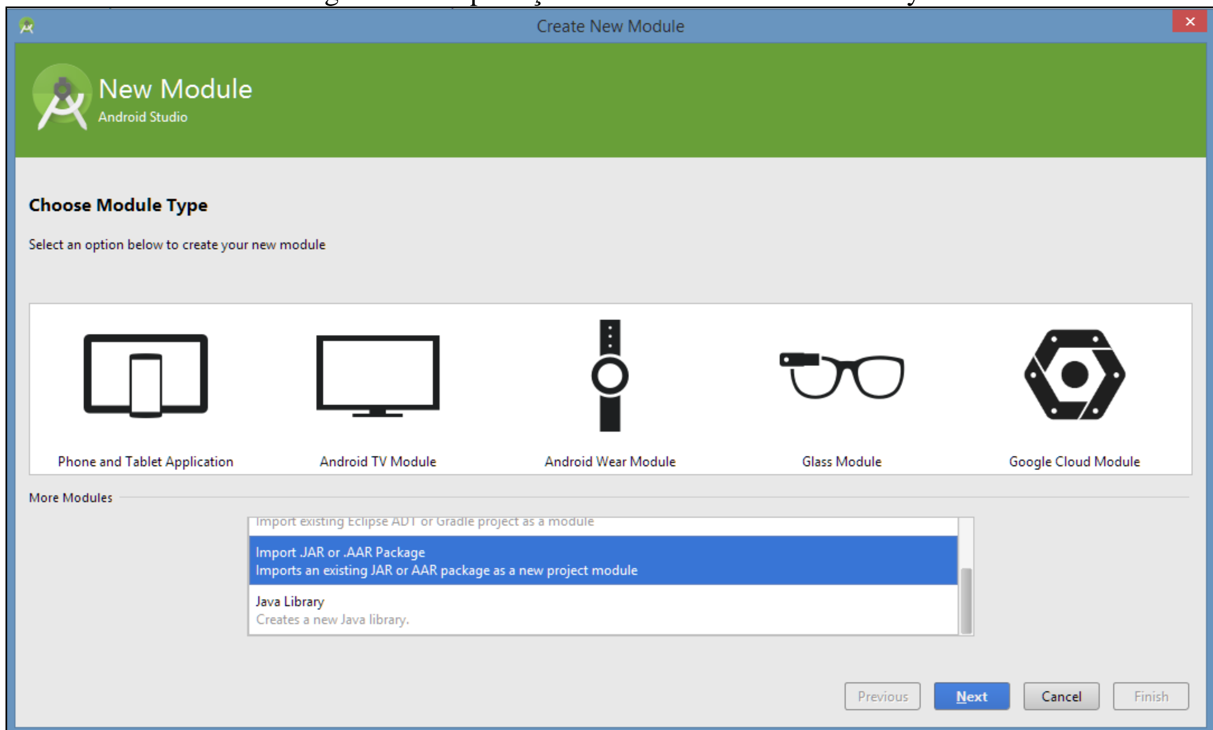
Ao iniciar um novo projeto visando a utilização da FurbRA-library, pode-se optar por utilizar a opção `No activity` existente no Android Studio (Figura 23) de maneira que estas serão definidas mais a frente com as classes fornecidas pela biblioteca.

Figura 23 - Tela de definição da activity inicial do projeto



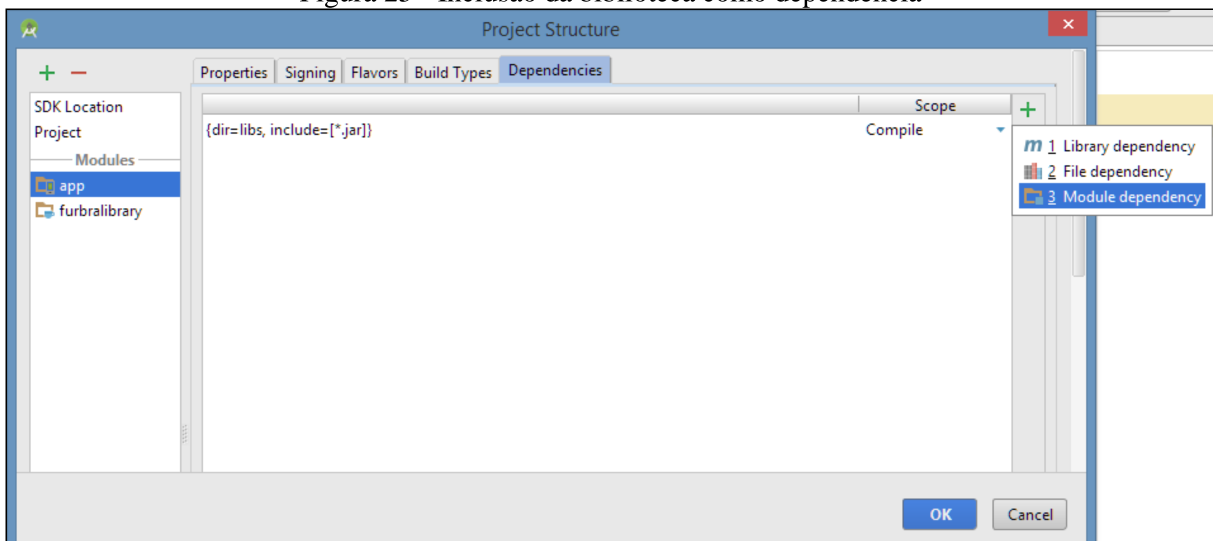
O próximo passo para a utilização da FurbRA-library é adicioná-la ao projeto. Para tanto, deve-se criar um novo módulo. Nas configurações do novo módulo o desenvolvedor deve escolher a opção `Import .JAR ou .AAR package` (Figura 24). Na tela seguinte é solicitado o caminho do arquivo a ser importado nela deve-se informar o caminho completo até o arquivo `furbralibrary.aar`.

Figura 24 - Importação da biblioteca FurbRA-library



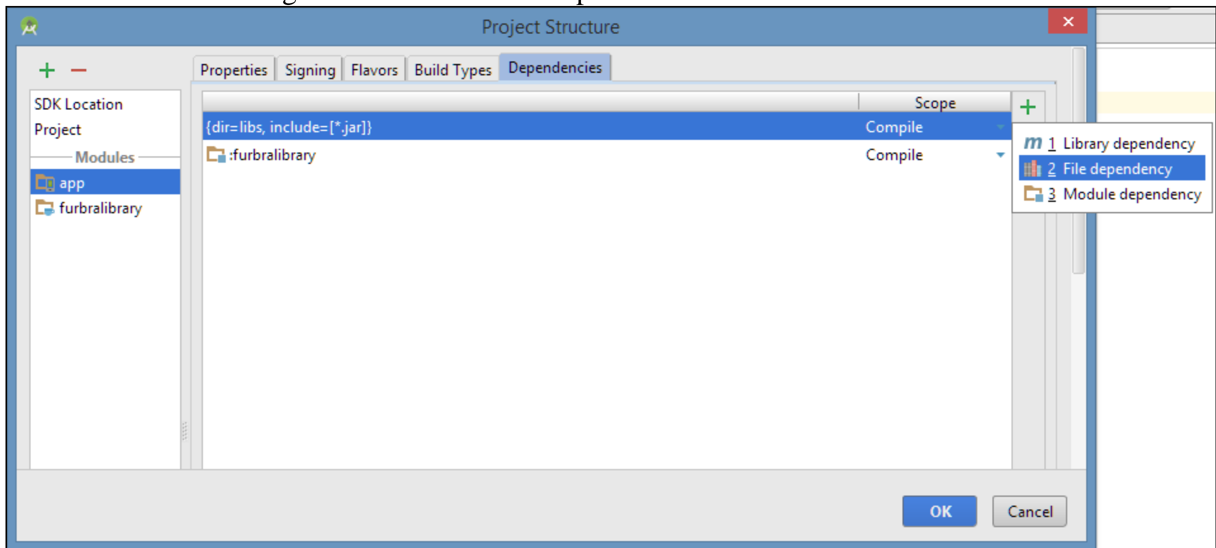

A seguir o desenvolvedor deve abrir a tela de `Project Structure`” da aplicação e na pasta `Dependencies` incluir o novo módulo importado como dependência ao projeto principal (Figura 25).

Figura 25 - Inclusão da biblioteca como dependência



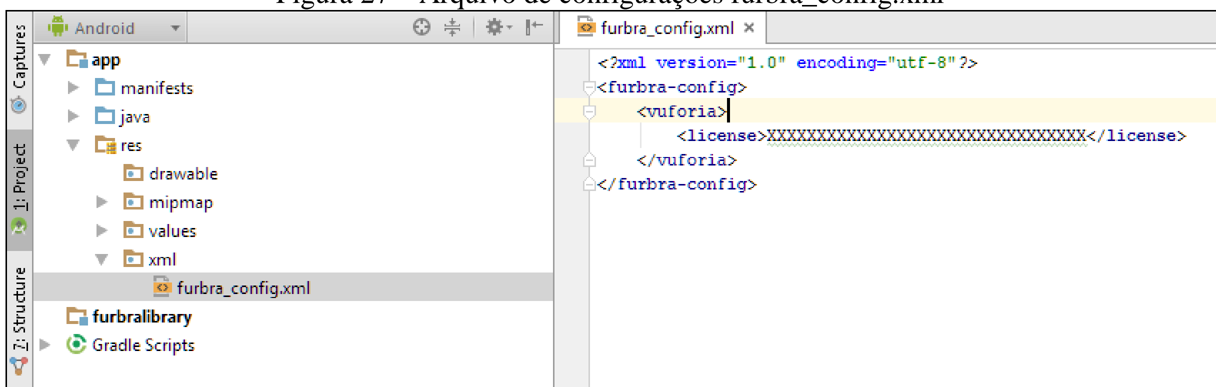
A seguir deve ser incluído como dependência do projeto a biblioteca Vuforia e as suas dependências. Esta inclusão se faz através do acréscimo dos arquivos Vuforia.jar (biblioteca Vuforia) e armebi.jar (dependência da biblioteca Vuforia), obtidos no portal de desenvolvedores da Vuforia, o desenvolvedor deve adicioná-los na pasta `libs` do projeto. Em seguida, novamente na tela `Project Structure` da aplicação, o mesmo deve adicionar estes arquivos como dependências (Figura 26).

Figura 26 - Inclusão das dependências da biblioteca Vuforia




Para finalizar, o desenvolvedor deve criar o arquivo de configurações da biblioteca, `furbra_config.xml`, e incluí-lo dentro do projeto principal na pasta `res/xml`. Dentro deste arquivo o desenvolvedor deve informar a licença obtida para utilização da biblioteca Vuforia (Figura 27).

Figura 27 – Arquivo de configurações furbra_config.xml




### 3.3.10.4 Utilizando a FurbRA-library

A FurbRA-library foi projetada de modo que o desenvolvedor não necessite de conhecimento avançado da plataforma Android para desenvolver jogos de realidade aumentada multijogadores com os recursos que a mesma oferece.

O primeiro passo para utilização da FurbRA-library é a definição das permissões necessárias à execução da biblioteca. As permissões são declaradas no arquivo `AndroidManifest.xml` e conforme o Quadro 57.

Quadro 57 - Permissões necessárias a utilização dos recursos da FurbRA-library

```
<uses-permission android:name="android.permission.CAMERA" />

<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />

<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"
/>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"
/>
<uses-permission android:name="android.permission.ACCESS_WIFI_STATE" />
```

O próximo passo é a criação da classe que herdará das *activities* existentes na biblioteca. Como exemplo serão demonstrados trechos da classe `MultiGameActivity` da aplicação teste Animais, que herda da classe `FurbRaTextRecognitionActivity` e possui em si a lógica quanto ao jogo *multiplayer*.

A primeira rotina demonstrada é o método `onCreate` (Quadro 58) utilizado para definir a estrutura de recursos que serão utilizados pelo jogo.

Quadro 58 - Implementação do método `onCreate` da classe `MultiGameActivity`

```
protected void onCreate(Bundle savedInstanceState) {
    //Configuração da tela para ocultação da barras de status
    this.requestWindowFeature(Window.FEATURE_NO_TITLE);
    this.getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
                              WindowManager.LayoutParams.FLAG_FULLSCREEN);

    super.onCreate(savedInstanceState);

    // Define que somente as palavras da lista adicional devem ser
    // consideradas
    setOnlyWordsInList(true);

    // Recupera a referência do jogador que será o oponente
    recoverPLayer();

    // Carrega os sons dos animais
    loadSounds();

    // Carrega as texturas dos animais
    loadTextures();

    // Define o listener de touch pra disparar a repodução do som
    setOnTouchListener(new View.OnTouchListener() {
        @Override
        public boolean onTouch(View v, MotionEvent event) {
            playAnimalSound();
            return true;
        }
    });

    // Cria a barra de progresso
    createProgressDialog();

    // Registra os listeners de recebimento de mensagens
    registerCommListeners();
}
```

Nesta rotina pode-se destacar três métodos que utilizam recursos oferecidos pela biblioteca FurbRA-library: `loadSounds`, `loadTextures` e `registerCommListeners`. O método `loadSounds` (Quadro 59) utiliza a classe `SoundBox` para realizar o carregamento assíncrono dos recursos sonoros que serão utilizados pela aplicação. O método `loadTextures` (Quadro 60) utiliza a classe utilitária `Textures` para realizar o carregamento de arquivos gráficos que serão utilizados na renderização. O método `registerCommListeners` (Quadro 61) utiliza a classe `LocalNetworkCommunication` para obter notificações referentes aos seguintes eventos:

a) perda de comunicação com o jogador adversário;
b) fim do carregamento inicial dos recursos do jogador adversário;
c) fim do turno do jogador adversário notificando quanto ao resultado.

Quadro 59 - Implementação do método loadSounds da classe MultiGameActivity

```
private void loadSounds() {
    //Carregamento dos sons
    SoundBox soundBox = SoundBox.getInstance();
    soundBox.addSound(APLAUSE_SOUND, R.raw.applause);
    soundBox.addSound(BUNNY_SOUND, R.raw.coelho);
    soundBox.addSound(COCK_SOUND, R.raw.galo);
    soundBox.addSound(CAT_SOUND, R.raw.gato);
    soundBox.addSound(DOG_SOUND, R.raw.cachorro);
    soundBox.addSound(MONKEY_SOUND, R.raw.macaco);
    soundBox.addSound(MOUSE_SOUND, R.raw.rato);
    soundBox.addSound(COW_SOUND, R.raw.vaca);
    soundBox.addSound(ZEBRA_SOUND, R.raw.zebra);
    soundBox.load(this);

    //Notificação da finalização;
    soundBox.setListener(new SoundBoxListener() {
        @Override
        public void resourceLoadingCompleted() {
            soundDone = true;
            checkForGameStart();
        }
    });
}
```

Quadro 60 - Implementação do método `loadTextures` da classe `MultiGameActivity`

```
private void loadTextures() {
    textures = new HashMap<>();
    textures.put("COELHO",
                Texture.loadTextureFromDrawable(R.drawable.coelho, this));
    textures.put("COELHO_LIBRA",
          Texture.loadTextureFromDrawable(R.drawable.coelho_libra, this));
    textures.put("CACHORRO",
              Texture.loadTextureFromDrawable(R.drawable.cachorro, this));
    textures.put("CACHORRO_LIBRA",
        Texture.loadTextureFromDrawable(R.drawable.cachorro_libra, this));
    textures.put("GALO",
                  Texture.loadTextureFromDrawable(R.drawable.galo, this));
    textures.put("GALO_LIBRA",
            Texture.loadTextureFromDrawable(R.drawable.galo_libra, this));
    textures.put("GATO",
                  Texture.loadTextureFromDrawable(R.drawable.gato, this));
    textures.put("GATO_LIBRA",
            Texture.loadTextureFromDrawable(R.drawable.gato_libra, this));
    textures.put("MACACO",
                Texture.loadTextureFromDrawable(R.drawable.macaco, this));
    textures.put("MACACO_LIBRA",
          Texture.loadTextureFromDrawable(R.drawable.macaco_libra, this));
    textures.put("RATO",
                  Texture.loadTextureFromDrawable(R.drawable.rato, this));
    textures.put("RATO_LIBRA",
            Texture.loadTextureFromDrawable(R.drawable.rato_libra, this));
    textures.put("VACA",
                  Texture.loadTextureFromDrawable(R.drawable.vaca, this));
    textures.put("VACA_LIBRA",
            Texture.loadTextureFromDrawable(R.drawable.vaca_libra, this));
    textures.put("ZEBRA",
                 Texture.loadTextureFromDrawable(R.drawable.zebra, this));
    textures.put("ZEBRA_LIBRA",
           Texture.loadTextureFromDrawable(R.drawable.zebra_libra, this));
    textures.put("GANHOU",
              Texture.loadTextureFromDrawable(R.drawable.parabens, this));
```

```java
    textures.put("PERDEU",
                Texture.loadTextureFromDrawable(R.drawable.perdeu, this));
    textures.put("EMPATE",
                Texture.loadTextureFromDrawable(R.drawable.empate, this));
    textures.put("AGUARDE",
               Texture.loadTextureFromDrawable(R.drawable.aguarde, this));
    textures.put("ACERTOU_LIBRA",
          Texture.loadTextureFromDrawable(R.drawable.acertou_libra, this));
}
```

Quadro 61 - Implementação do método `registerCommListeners` da classe `MultiGameActivity`

```java
private void registerCommListeners() {
    final LocalNetworkCommunication instance =
                                LocalNetworkCommunication.getInstance();

    instance.registerLocalExternalServiceDiscoveryListener(
    new LocalExternalServiceDiscoveryListener() {
        @Override
        public void serviceDiscovered(LanService service) {}

        @Override
        public void serviceLost(LanService service) {
            if (service.equals(player.getLanService())) {
                //Se o servico caiu nao pode se continuar o jogo;
                showToast("O amiguinho foi desconectado.");
                new Timer().schedule(new TimerTask() {
                    @Override
                    public void run() {
                        finish();
                    }
                }, 5000);
            }
        }
    });

    instance.registerLocalNetworkMessageReceivedListener(
       AllSetToStartMessage.class,
       new LocalNetworkMessageReceivedListener() {
        @Override
        public void messageReceived(LanService service, Message message) {
            otherPlayerDone = true;
            checkForGameStart();
        }
    });

    instance.registerLocalNetworkMessageReceivedListener(
       ResultNotificationMessage.class,
       new LocalNetworkMessageReceivedListener() {
        @Override
        public void messageReceived(LanService service, Message message) {
            ResultNotificationMessage msg =
                                (ResultNotificationMessage) message;
            String palavra = msg.getPalavra();
            boolean acertou = msg.isAcertou();
            if (acertou) {
                otherUserScore++;
            }

            expectedWords.remove(palavra.toLowerCase());

            currentPlayerTurn = true;
            nextWord();
```

```
        }
    });
}
```

O próximo método demonstrado é o `trackerFound` (Quadro 62), utilizado para detectar se o usuário formou corretamente a palavra esperada. Caso a palavra detectada seja a palavra esperada, a rotina executa o método `wordFinded`.

Quadro 62 - Implementação do método `trackerFound` da classe `MultiGameActivity`

```
public void trackerFound(TrackableResult trackableResult, float[] matrix){
    if (!trackableResult.isOfType(WordResult.getClassType()) ||
        !this.searching) {
        return;
    }

    WordResult wordResult = (WordResult) trackableResult;
    Word word = (Word) wordResult.getTrackable();

    if (word.getStringU().equalsIgnoreCase(this.currentWord)) {
        wordFinded();
    }
}
```

O método `wordFinded` (Quadro 63), executado pelo método de detecção acima, notifica o usuário quanto ao acerto, através de som e imagem, e envia mensagem ao adversário notificando quanto ao final da rodada e quanto ao acerto do usuário.

Quadro 63 - Implementação do método `wordFinded` da classe `MultiGameActivity`

```
private void wordFinded() {
    //cancela o timer da rodada
    timer.cancel();
    //Atualiza o score
    currentUserScore++;
    //Inativa a busca
    this.searching = false;
    //Mostra a barra de status como verde.
    topBarObject.setColor(0, 1f, 0, 1f);
    //Toca o som de aplausos
    SoundBox.getInstance().play(APLAUSE_SOUND);

    //Deixa a mensagem de parabens por 2 segundos
    new Timer().schedule(new TimerTask() {
        @Override
        public void run() {
            searching = true;

            //Oculta a barra de status.
            topBarObject.setColor(0, 1f, 0, 0f);

            try {
                //Atualiza a flag da rodada
                currentPlayerTurn = false;
                //Envia a mensagem quanto ao fim da rodada
                LocalNetworkCommunication.getInstance()
                            .sendMessage(player.getLanService(),
                             new ResultNotificationMessage(true,
                             currentWord));
                if (expectedWords.size() == 0) {
                    //Caso nao haja mais palavras finaliza o jogo
                    finishGame();
                }
            } catch (IOException e) {
                new RuntimeException(e);
            }
        }
    }, 2000);
}
```

## 3.4 RESULTADOS E DISCUSSÕES

Os resultados obtidos pela FurbRA-library foram medidos através de testes experimentais da sua capacidade de renderizar cenas e na capacidade de envio/recebimento de mensagens. Todos os testes foram realizados iniciando a aplicação a partir do Android Studio e observando a sua execução no dispositivo Nexus 7.

Acredita-se que o fator principal que pode afetar o desempenho gráfico da FurbRA-library, seria a quantidade de objetos virtuais sendo renderizados. O método utilizado para medir o desempenho de renderização foi a contagem do número FPS apresentados pela aplicação desenvolvida utilizando a FurbRA-library. Para tanto, foram realizadas medições da taxa de FPS com diferentes quantidades de objetos sendo desenhados ao mesmo tempo. As medições foram realizadas com a imagem da câmera sendo renderizada ao fundo, com os

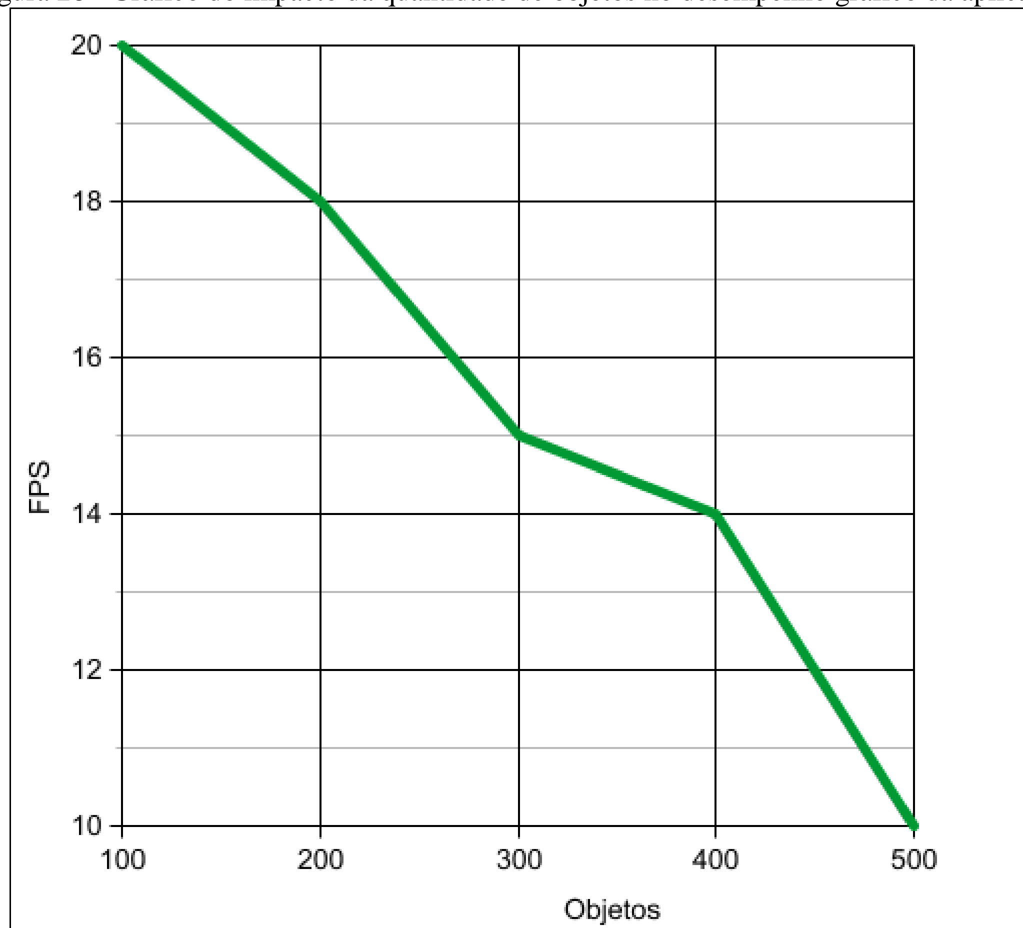
objetos gráficos sendo renderizados a frente, com a transparência habilitada e utilizando objetos simples, formados por 6 vértices. Os resultados obtidos são apresentados na Tabela 1.

Tabela 1 - Medição da taxa de FPS na renderização de objetos

| Quantidade de objetos | Média de FPS |
|---|---|
| 100 | 20 |
| 200 | 18 |
| 300 | 15 |
| 400 | 13 |
| 500 | 10 |

Com base nos resultados obtidos, é possível observar que o desempenho da aplicação se mostrou sensível a quantidade de objetos sendo renderizados. É interessante ressaltar que os testes foram realizados ainda com câmera sendo renderizada ao fundo, o que resulta em uma grande quantidade de processamento gráfico sendo executado. Optou-se por realizar os testes dessa maneira por acreditar-se que esse é o padrão nativo das aplicações de RA. A Figura 28 mostra o gráfico gerado com base nas medições, com o desempenho obtido através da medição do FPS na vertical e a quantidade de objetos na horizontal.

Figura 28 - Gráfico do impacto da quantidade de objetos no desempenho gráfico da aplicação




No segundo conjunto de testes foi verificado o desempenho nos métodos de envio e recebimento das mensagens através da FurbRA-library. Para tanto foi montado um aplicativo

de testes que foi carregado nos tablets Nexus 7 e Samsung Galaxy Tab. Um dos dispositivos serviu como o criador das mensagens, lançando a mensagem ao segundo dispositivo que retornaria a mesma ao remetente. Como mensagem foi criada uma classe que herdava da classe `Message`, contendo como único atributo um identificador numérico inteiro. A mensagem binária criada a partir da instância desta classe continha em média 180 bytes. Como os testes de comunicação envolvem um número considerado de variáveis, entre as principais podendo citar a rede WiFi, escolheu-se por realizar os testes em três turnos, executados em momentos distintos, e variando os aparelhos entre as funções de envio e retorno das mensagens.

Na Tabela 2 são apresentados os tempos médios obtidos entre o envio e o recebimento das mensagens nos três turnos de testes.

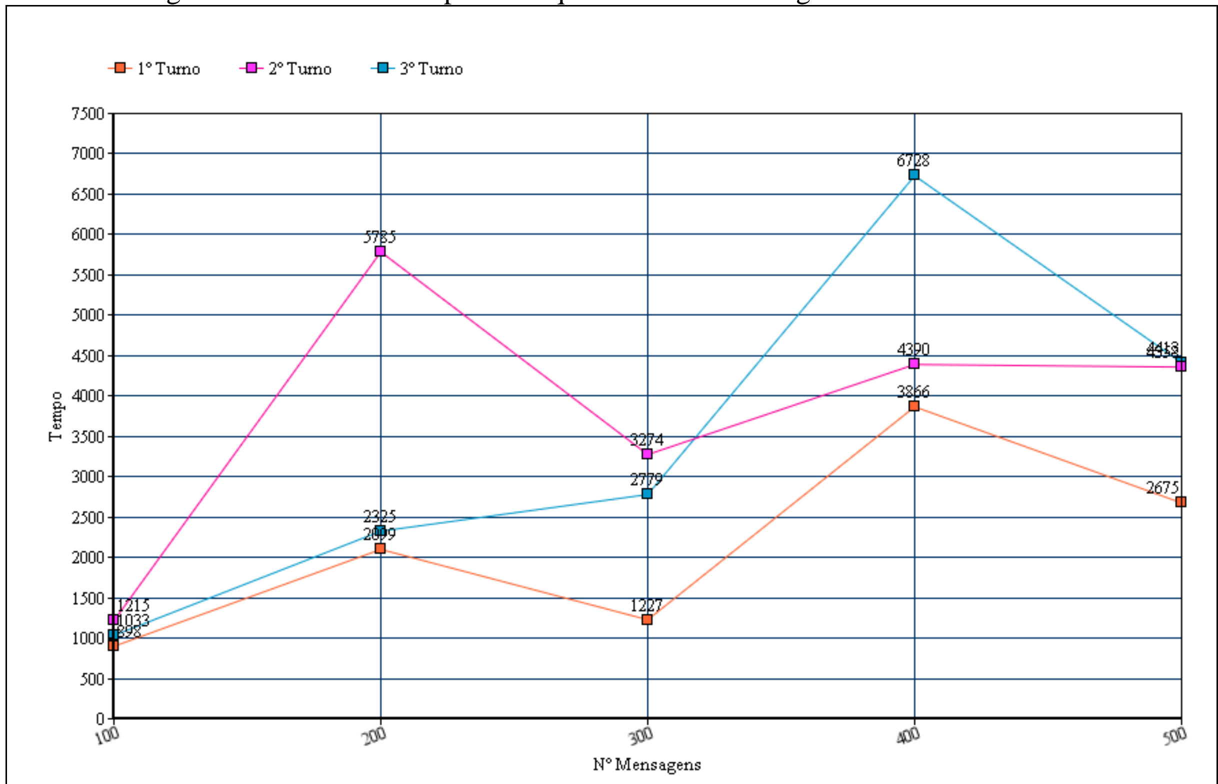
Tabela 2 - Medição de tempo entre o envio e recebimento de mensagens

| Quantidade de mensagens enviadas | Tempo médio entre o envio e o recebimento | | |
|---|---|---|---|
| | 1º Turno | 2º Turno | 3º Turno |
| 100 | 898 ms | 1215 ms | 1033 ms |
| 200 | 2099 ms | 5785 ms | 2325 ms |
| 300 | 1227 ms | 3274 ms | 2779 ms |
| 400 | 3866 ms | 4390 ms | 6728 ms |
| 500 | 2675 ms | 4358 ms | 4413 ms |

Com base nos resultados obtidos, desconsiderando flutuações na rede, é possível observar que o desempenho da aplicação se mostrou sensível à quantidade de objetos sendo enviados/recebidos. Analisando os dados, supõe-se que esta sensibilidade seja em decorrência do processo de comunicação utilizado. A serialização e deserialização de objetos resultava em uma grande quantidade de objetos sendo criados em memória. Assim que tais objetos ocupavam uma determina quantidade de memória o sistema operacional realizava a limpeza através do *garbage collector*. A esse processo de limpeza, concorrente ao processo principal, responsabiliza-se o aumento no tempo médio de envio/recebimento da mensagem. É importante ressaltar que esta suposição é baseada somente nos dados apresentados, dos quais pode-se obter diferentes interpretações. Para se chegar a um parecer concreto seria necessário realizar um aprofundamento e investigação quanto aos fatores envolvidos nesses resultados.

A Figura 29 mostra o gráfico gerado com base nas medições, com o desempenho obtido através da medição de tempo da comunicação na vertical e a quantidade de mensagens na horizontal.

Figura 29 - Gráfico do impacto da quantidade de mensagens enviadas/recebidas




### 3.4.1 Dificuldades encontradas

A principal dificuldade encontrada durante o desenvolvimento do trabalho foi quanto ao uso da especificação NSD. Esta apresentou problemas nos testes realizados no *tablet* Samsung Galaxy Tab, onde foi verificado uma sequência de problemas que normalmente culminavam na reinicialização do dispositivo, fato que não ocorria no *tablet* Nexus 7. Pela falta de dispositivos disponíveis para testes não foi possível aprofundar os testes e verificar se estes problemas ocorriam em relação ao aparelho ou a versão da plataforma.

Também em relação à especificação NSD foi possível verificar que até mesmo nas versões mais recentes da plataforma a especificação não funciona em sua totalidade. Um exemplo que pode ser observado nesse sentido é quanto a atributos extras que podem ser inseridos na criação do serviço NSD. Embora a plataforma possua métodos específicos para este fim os mesmos não executam sua finalidade proposta.

### 3.4.2 Comparativo entre a biblioteca desenvolvida e seus correlatos

Nessa seção é apresentada uma comparação entre as principais características da biblioteca FurbRA-library com as identificadas nos trabalhos definidos como correlatos do mesmo (Quadro 64).

Quadro 64 - Comparação com os trabalhos correlatos

| Característica | Vasselai (2010) | ARToolKit | Unreal Engine | FurbRA-library |
|---|---|---|---|---|
| Criação de RA baseada em marcadores | | X | | X |
| Identificação simultânea de marcadores | | X | | X |
| Criação de RA baseada em sensores | X | | | |
| Identificação de textos | | | | X |
| Permite comunicação | | | X | X |
| Permite reprodução de arquivos sonoros | | | X | X |
| Renderização de objetos gráficos | | | X | X |

Independente da diferença de propósito dos trabalhos correlatos com a biblioteca desenvolvida, a FurbRA-library apresenta a maioria das características comparadas.

O trabalho de Vasselai (2010) destaca-se dos demais por tratar-se de um trabalho exploratório das possibilidades da plataforma Android para o desenvolvimento de aplicativos de RA, mas que foi base para a verificação da estrutura necessária para o desenvolvimento deste trabalho.

A biblioteca ARToolKit é voltada exclusivamente à criação de RA baseada em marcadores visuais de modo que características voltadas a comunicação ou recursos multimídia não são encontrados na mesma. Também não são oferecidos pela biblioteca recursos de identificação de textos.

A *engine* Unreal Engine é a que mais se aproxima da biblioteca desenvolvida sendo que em ambas existe o propósito de auxiliar no desenvolvimento de jogos, embora a *engine* seja mais abrangente enquanto a biblioteca desenvolvida possui foco na área de jogos de RA. A Unreal Engine, assim como a FurbRA-library, também oferece serviços como a de reprodução de arquivos sonoros, comunicação e renderização de objetos gráficos.

# 4 CONCLUSÕES

O presente trabalho apresentou o projeto e desenvolvimento de uma biblioteca para desenvolvimento de jogos multijogadores de Realidade Aumentada para Android, que permite a construção de jogos que seguem o conceito de RA aplicando as três propriedades básicas definidas por Azuma (1997) e apresentadas na seção 2.2.

A biblioteca desenvolvida obteve êxito na disponibilização de um conjunto de funções e recursos de forma a facilitar a criação da Realidade Aumentada baseada em marcadores, sendo oferecido suporte a dois tipos de marcadores.

Outro objetivo alcançado foi a disponibilização de um conjunto de funções e recursos de forma a facilitar o gerenciamento do áudio. O objetivo proposto quanto a um conjunto de funções e recursos de forma a facilitar a comunicação local entre os aparelhos através da especificação de rede WiFi também foi alcançado com êxito.

Foi apresentada também, um conjunto de funções e recursos de forma a facilitar a utilização da biblioteca OpenGL ES necessária no desenvolvimento gráfico dos jogos de Realidade Aumentada.

Embora foi possível atingir os objetivos propostos, certos objetivos inicias foram alterados durante a execução. Inicialmente a proposta era de utilizar a bússola, o GPS e o acelerômetro para detectar mudanças na situação do ambiente do dispositivo. Porém, tais sensores não foram utilizados tendo em vista que a abordagem que foi seguida durante a execução do trabalho foi do desenvolvimento da Realidade Aumentada a partir de marcadores visuais.

Em sua concepção inicial, a FurbRA-library também deveria fornecer meios para a comunicação através da especificação bluetooth. Este requisito não foi implementado, pois no decorrer do desenvolvimento da FurbRA-library verificou-se que a possibilidade de comunicação através das especificações NSD e da rede WiFi produzem melhores resultados quanto a mobilidade e desempenho.

Outro ponto inicial proposto, e que foi alterado durante o desenvolvimento deste, foi o gerenciamento de arquivos. Na concepção inicial a biblioteca deveria fornecer mecanismos facilitadores para a manipulação de arquivos. Este requisito acabou não sendo implementado, pois no decorrer do desenvolvimento da FurbRA-library verificou-se que não se tratava de uma necessidade primária.

Os resultados finais foram satisfatórios, pois foi possível construir um jogo de Realidade Aumentada multijogadores utilizando os recursos disponibilizados pela FurbRA-library, que segue as três propriedades básicas definidas por Azuma (1997).

Por fim, a biblioteca desenvolvida possui uma estrutura simples de utilizar, modular, sem acoplamento nos módulos desenvolvidos e que permite a desenvolvedores sem conhecimento profundo da plataforma Android desenvolverem jogos que implementam o conceito de Realidade Aumentada e comunicação local.

## 4.1 EXTENSÕES

Durante o desenvolvimento foram verificadas algumas possíveis melhorias que não foram contempladas neste trabalho.

A gama de tipos de marcadores oferecidos pela biblioteca Vuforia vai além dos tipos implementados. Como melhoria sugere-se a implementação dos demais tipos de marcadores a fim de aumentar as possibilidades de desenvolvimento da aplicação final.

Em relação a renderização, sugere-se a implementação da importação de objetos gráficos através de arquivos de extensão `obj` de maneira a renderizar objetos mais complexos sem ter que lidar com os vértices em código.

Outra possível melhoria seria a adaptação do código para utilização em dispositivos *eyewear,* dos quais a biblioteca Vuforia oferece suporte como é o caso, atualmente, dos dispositivos Epson Moverio BT-200, ODG R-6 e Samsung GearVR Innovator Edition.

Também se sugere uma investigação quanto aos fatores envolvidos nos resultados dos testes de comunicação, que mostraram variações dos quais não foi possível obter um parecer assertivo.

## REFERÊNCIAS

AZUMA, Ronald T. A Survey of augmented reality. **Presence**: Teleoperators and Virtual Environments. [S.l.], v. 6, n. 4, p. 355-385. ago. 1997. Disponível em: <http://www.cs.unc.edu/~azuma/ARpresence.pdf>. Acesso em: 26 mar. 2014.

AZUMA, Ronald T. et al. **Recent advances in augmented reality**, [S.l], 2001. Disponível em: <http://www.cc.gatech.edu/~blair/papers/ARsurveyCGA.pdf>. Acesso em: 23 mar. 2014.

BUCHMANN, Volkert et al. **FingARtips** – gesture based direct manipulation in augmented reality. [S.l.], 2004. Disponível em: <http://www.cosc.canterbury.ac.nz/andrew.cockburn/papers/fingartips.pdf>. Acesso em: 20 mar. 2014.

DAQRI. **ARToolKit Documentation** [ARToolkit]. [S.l.], 2015a. Disponível em: <http://artoolkit.org/documentation/>. Acesso em: 09 jul. 2015.

DAQRI. **ARToolKit Feature Comparison** [ARToolkit]. [S.l.], 2015b. Disponível em: <http://artoolkit.org/documentation/doku.php?id=1_Getting_Started:about_feature_comparison>. Acesso em: 09 jul. 2015.

DAQRI. **Open Source Augmented Reality SDK** | ARToolKit.org. [S.l.], 2015c. Disponível em: <http://artoolkit.org/>. Acesso em: 09 jul. 2015.

EPICGAMES. **About Unreal Engine 4**. [S.l.], 2015a. Disponível em: <https://www.unrealengine.com/unreal-engine-4>. Acesso em: 10 jul. 2015.

EPICGAMES. **Custom Licensing**. [S.l.], 2015b. Disponível em: <https://www.unrealengine.com/custom-licensing>. Acesso em: 10 jul. 2015.

EPICGAMES. **What is Unreal Engine 4**. [S.l.], 2015c. Disponível em: <https://www.unrealengine.com/what-is-unreal-engine-4>. Acesso em: 10 jul. 2015.

F2Z. **Pandemic – Coop ::** Zman Games. [S.l.], 2014. Disponível em: <http://zmangames.com/product-details.php?id=1246>. Acesso em: 08 abr. 2014.

GOOGLE. **Activities** | Android Developers. [S.l.], 2015a. Disponível em: <http://developer.android.com/about/index.html>. Acesso em: 18 abr. 2015a.

______. **Android, the world's most popular mobile platform**. [S.l.], 2014a. Disponível em: <http://developer.android.com/about/index.html>. Acesso em: 15 mar. 2014a.

______. **Bluetooth** | Android developers. [S.l.], 2014b. Disponível em: <http://developer.android.com/guide/topics/connectivity/bluetooth.html>. Acesso em: 15 mar. 2014b.

______. **Camera** | Android developers. [S.l.], 2014c. Disponível em: <http://developer.android.com/guide/topics/media/camera.html>. Acesso em: 15 mar. 2014c.

______. **Connectivity** | Android developers. [S.l.], 2014d. Disponível em: <http://developer.android.com/guide/topics/connectivity/index.html>. Acesso em: 15 mar. 2014d.

______. **Dashboards** | Android developers. [S.l.], 2014e. Disponível em: <http://developer.android.com/about/dashboards/index.html>. Acesso em: 15 mar. 2014e.

______. **Media playback** | Android developers. [S.l.], 2014f. Disponível em: <http://developer.android.com/guide/topics/media/mediaplayer.html>. Acesso em: 15 mar. 2014f.

______. **Near field communication** | Android developers. [S.l.], 2014g. Disponível em: <http://developer.android.com/guide/topics/connectivity/nfc/index.html>. Acesso em: 15 mar. 2014g.

______. **OpenGL ES** | Android developers. [S.l.], 2014h. Disponível em: <http://developer.android.com/guide/topics/graphics/opengl.html>. Acesso em: 15 mar. 2014h.

______. **Pausing and Resuming an Activity** | Android developers. [S.l.], 2015b. Disponível em: < http://developer.android.com/training/basics/activity-lifecycle/pausing.html>. Acesso em: 27 mai. 2015b.

______. **Sensors overview** | Android developers. [S.l.], 2014i. Disponível em: <http://developer.android.com/guide/topics/sensors/sensors_overview.html>. Acesso em: 15 mar. 2014i.

______. **Using Network Service Discovery** | Android Developers. [S.l.], 2015c. Disponivel em: <http://developer.android.com/training/connect-devices-wirelessly/nsd.html>. Acesso em: 29 mai. 2015c.

HANDRAHAN, Matthew. **League of Legends 2013 revenue topped $600m** – report | GameIndustry International. [S.l.], 2014. Disponível em: <http://www.gamesindustry.biz/articles/2014-01-20-league-of-legends-2013-revenue-toppedUSD600m-report>. Acesso em: 08 abr. 2014.

INT13. **ARDefender.** [S.l.], 2012. Disponível em: <http://www.ardefender.com/>. Acesso em: 08 abr. 2014.

KARCH, Marziah. **What is Android** – Google open sorce plataform for mobile phones. [S.l.], 2014. Disponível em: <http://google.about.com/od/socialtoolsfromgoogle/p/android_what_is.htm/>. Acesso em: 16 mar. 2014.

LAMB, Philip, **ARToolKit Home Page.** [S.l.], 2010. Disponivel em: <http://www.hitl.washington.edu/artoolkit/>. Acesso em: 09 jul. 2015.

LOWOOD, Henry. Videogames in computer space: the complex history of pong. **IEEE Annals of the History of Computing, Stanford**, v. 31, n. 3, p. 05-19, jul. 2009.

MARKETSANDMARKETS. **MarketsandMarkets:** global augmented reality (AR) market worth $5151.74 million by 2016. [S.l.], 2014. Disponível em: <http://www.marketsandmarkets.com/PressReleases/ar-market.asp>. Acesso em: 17 mar. 2014.

QUALCOMM. **Augmented reality (Vuforia)**. [S.l.], 2014a. Disponível em: <https://developer.qualcomm.com/mobile-development/add-advanced-features/augmentedreality-vuforia>. Acesso em: 20 mar. 2014.

______. **Extended tracking** | Vuforia developer portal. [S.l.], 2014b. Disponível em: <https://developer.vuforia.com/resources/dev-guide/extended-tracking>. Acesso em: 20 mar. 2014.

______. **Frame Markers** | Vuforia Library Prod. [S.l.], 2015a. Disponível em: <https://developer.vuforia.com/library/articles/Training/Frame-Markers-Guide>. Acesso em: 01 mai. 2015.

______. **Vuforia Developer Portal**. [S.l.], 2015b. Disponível em: <https://developer.vuforia.com >. Acesso em: 11 abr. 2015.

REUTERS. **FACTBOX - A look at the $66 billion video-games industry**. [S.l.], 2013. Disponível em: <http://in.reuters.com/article/2013/06/10/gameshow-eidINDEE9590DW20130610>. Acesso em: 08 abr. 2014.

SHENANDOAH. **Drive on Moscow**. [S.l.], 2014. Disponível em: <http://www.shenandoahstudio.com/dom/>. Acesso em: 08 abr. 2014.

STATISTA. **World of Warcraft subscriber number 2005-2013** | Statistic. [S.l.], 2014. Disponível em: <http://www.statista.com/statistics/276601/number-of-world-of-warcraftsubscribers-by-quarter/>. Acesso em: 08 abr. 2014.

VASSELAI, Gabriela Tinti. **Um estudo sobre realidade aumentada para a plataforma android**. 2010. 102 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

YEUNG, Ken. **Google glass named one of time's 2012 best inventions**. [S.l.], 2012. Disponível em: <http://thenextweb.com/gadgets/2012/11/02/google-glass-best-invention2012/>. Acesso em: 17 mar. 2014.

**APÊNDICE A – Gabarito paras as peças do jogo Animais**

A Figura 30 apresenta o gabarito para as peças do jogo Animais.

Figura 30 - Gabarito paras as peças do jogo Animais

## APÊNDICE B – Formulário de avaliação da experiência de usuário da aplicação Animais

O Quadro 65 apresenta o formulário de avaliação de usabilidade da aplicação teste Animais.

Quadro 65- Formulário de avaliação da experiência de usuário da aplicação Animais

# Experiência de Uso Aplicativo Educacional

BIBLIOTECA PARA DESENVOLVIMENTO DE JOGOS MULTIJOGADORES DE REALIDADE AUMENTADA PARA ANDROID
Trabalho de Conclusão de Curso de Bacharelado em Ciências da Computação
Título: Jogo educativo "Animais"
Acadêmico: Diego André Lira
Orientador: Dalton Solano dos Reis

Resumo: este trabalho apresenta a implementação de uma biblioteca para a plataforma Android que possibilita a criação de jogos multijogadores de realidade aumentada. Com o uso desta biblioteca foi criado o jogo educativo "Animais". Neste jogo, voltado a crianças em processo de alfabetização, as crianças utilizam letras para montar nomes de animais que serão apresentados nos dispositivos. A cada resposta correta a criança é recompensada com estímulos positivos como sons e cores. O jogo também utiliza a linguagem brasileira de sinais (libras) de maneira a estimular a criança ao aprendizado desta, estimulando assim a inclusão de pessoas com deficiência auditiva.

*Obrigatório

1. Como você classifica a usabilidade da aplicação? *

*Marcar apenas uma opção.*

( ) Intuitiva
( ) Atende as necessidades
( ) Boa
( ) Outro: ____________________________________________

2. Você considera o conjunto de recursos apresentados (textos, sons e imagens) atrativos para crianças em idade de alfabetização? *
*Marcar apenas uma opção.*
( ) Pouco atrativos
( ) Atende a necessidade
( ) Atrativos
( ) Outro: ____________________________________________

3. Você considera a aplicação apresentada como útil para o processo de alfabetização de crianças?
*Marcar apenas uma opção.*
( ) Pouco útil
( ) Atende a necessidade
( ) Útil
( ) Outro: ____________________________________________

4. Durante a utilização do jogo, foi possível realizar o jogo com mais de uma pessoa através da opção de menu "Com Amiguinhos" *
*Marcar apenas uma opção.*
( ) Sim

( ) Não

5. Com base nas respostas anteriores, quais seriam os principais "Pontos Positivos" no uso do aplicativo? *

______________________________________________________________________

______________________________________________________________________

6. Com base nas respostas anteriores, quais seriam os principais "Pontos Negativos" no uso do aplicativo? *

______________________________________________________________________

______________________________________________________________________

7. Com base nas respostas anteriores, quais seriam as suas "Sugestões" de melhoria para o aplicativo?

______________________________________________________________________

______________________________________________________________________