CENTRO UNIVERSITÁRIO DE BRASÍLIA - UniCEUB
CURSO DE ENGENHARIA DE COMPUTAÇÃO

EMERSON OLIVEIRA DA HORA

TRANSMISSÃO DE ÁUDIO EM REDES SEM FIO E DISPOSITIVOS MÓVEIS

Orientador: Prof. MsC Francisco Javier de Obaldía

Brasília
Dezembro, 2012

EMERSON OLIVEIRA DA HORA

TRANSMISSÃO DE ÁUDIO EM REDES SEM FIO E DISPOSITIVOS MÓVEIS

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.

Orientador: Prof. MsC Francisco Javier de Obaldía

Brasília

Dezembro, 2012

EMERSON OLIVEIRA DA HORA

TRANSMISSÃO DE ÁUDIO VIA STREAMING EM REDES SEM FIO E DISPOSITIVOS MÓVEIS

Trabalho apresentado ao Centro Universitário de Brasília (UniCEUB) como pré-requisito para a obtenção de Certificado de Conclusão de Curso de Engenharia de Computação.
Orientador: Prof. MsC Francisco Javier de Obaldía

Este Trabalho foi julgado adequado para a obtenção do Título de Engenheiro de Computação, e aprovado em sua forma final pela Faculdade de Tecnologia e Ciências Sociais Aplicadas - FATECS.

_____

Prof. Abiezer Amarilia Fernandes
Coordenador do Curso

Banca Examinadora:

_____

Prof. Francisco Javier de Obaldía, Mestre.
Orientador

_____

Prof. Luís Cláudio Lopes de Araújo, Mestre.
Uniceub

_____

Prof. nome, titulação.
Uniceub

_____

Prof. Fabiano D'Oliveira, Mestre.
Uniceub

AGRADECIMENTOS

Agradeço a todos que contribuíram de alguma forma com esse projeto e a todos que me auxiliaram no decorrer do curso de engenharia. Especialmente os seguintes:

Matheus Oliveira da Hora (Irmão)
Valdeneide Pinto Oliveira (Mãe)
Elias Conceição da Hora (Pai)
Prof. Javier
Prof. Luciano
Caio de Bem
Ana Gabriela
Manuella Thereza
Jefferson Santos
Lucas Rehem
Felipe Galiza
Diogo Holanda
Georges Elias Azar
Alexandro Coutinho
Lucas Eurípedes Oliveira

SUMÁRIO

RESUMO

Este trabalho apresenta uma solução para reprodução de músicas através de uma rede local, utilizando o computador pessoal e um *smartphone* com sistema operacional *android*. O produto gerado é um sistema no modelo cliente servidor, onde as músicas localizadas no servidor, o computador pessoal, serão transmitidas para o cliente, o *smartphone*, através do protocolo HTTP, sem a necessidade de armazenamento definitivo da música no celular, ela sendo armazenada apenas em buffer para a reprodução imediata. O sistema é composto de dois programas, o primeiro no servidor, uma aplicação desenvolvida em C#, responsável por gerenciar as músicas disponíveis para reprodução pelo cliente, e a segunda no cliente, desenvolvida em Java, responsável pela reprodução das músicas.

O objetivo desse trabalho é o desenvolvimento desse sistema, de forma que seja possível a conexão de pelo menos um cliente ao servidor e que a reprodução de músicas atinja um nível satisfatório em uma rede local sem fio.

Palavras Chave: HTTP, RTP, Rede sem fio, C#, Java, Android, Música

ABSTRACT

This paper presents a solution for music playback over a local network using a personal computer and a smartphone with android operating system. The product generated is a system in client server model, where the songs are located on the server, personal computer, will be transmitted to the client, smartphone, via HTTP, without the need for permanent storage of music on your phone, it is stored only buffer for immediate playback. The system consists of two programs, the first server, an application developed in C #, responsible for managing the songs available for playback by the client and the second client, developed in Java, responsible for the reproduction of music.

The aim of this work is the development of this system, so that it is possible to connect at least one client to the server and music playback reaches a satisfactory level in a wireless LAN.

Key words: HTTP, RTP, Wi-Fi network, C#, Java, Android, Music

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ABREVEATURAS

API	Application Programming Interface
CLR	Common Language Runtime
HTML	HyperText Markup Language
HTTP	Hiper Text Transport Protocol
ISO	International Standardization Organization
MIME	Multipurpose Internet Mail Extensions
MPEG	Motion Picture Expert Group
PCM	Pulse Code Modulation
RTCP	RTP control protocol
RTP	Real-time Transport Protocol
SOAP	Simple Object Access Protocol
URL	Uniform Resource Locator
WWW	World Wide Web
WCF	Windows Comunication Foundation
XML	eXtensible Markup Language

CAPÍTULO 1 – INTRODUÇÃO

Neste projeto, para a criação do sistema de *streaming* foram necessários diversos conceitos tanto do nível de programação, como conceitos de redes e da estrutura do objeto a ser transmitido: arquivos no formato MP3. A utilização dessas tecnologias permitiu a criação do sistema integrado gerado no projeto.

O MP3 é um padrão de codificação de sinais de áudio e vídeo, com o objetivo de facilitar o armazenamento de sinais interativos de áudio em mídias digitais. O padrão foi desenvolvido e proposto para a codificação de sinais de áudio estéreo em 2 x 192 bits/s, 2 x 128 bits/s e 2 x 64 bits/s, em 1992. A abreviação MP3 foi introduzida para substituir o nome extenso do padrão de codificação MPEG-1, camada 3. (MUSMANN, 2003)

A tecnologia Wi-Fi permite a conexão de dispositivos através de redes sem fio. Ela utiliza o padrão IEEE 802.11 que se divide em vários subpadrões, entre eles estão o 802.11b que utiliza as faixas de frequência de 2,4GHz a 2,485GHz e permite uma taxa de dados de até 11 Mbps, a 802.11a que utiliza as faixas de 5,1GHz a 5,8GHz e permite uma taxa de até 54Mbps e a 802.11g que utiliza as faixas de 2,4Ghz a 2,485GHz e permite uma taxa de até 54Mbps. (KUROSE e ROSS, 2009)

O *streaming*, ou fluxo de mídia, é uma forma de distribuir informação de mídia pela internet através de pacotes, com o objetivo de reprodução da mídia, seja áudio ou vídeo, ao mesmo tempo em que a mídia seja enviada, sem a necessidade do armazenamento da mídia pelo lado do usuário, apenas em questão de buffer de reprodução, ou seja, o arquivo de mídia é armazenado apenas durante o processo de reprodução, onde ao término da reprodução o arquivo é apagado. (FARIA, 2012)

O HTTP é um protocolo usado para comunicação no World Wide Web (WWW), seja através de navegadores, servidores ou outras aplicações. O HTTP é utilizado para transferir a maior parte das informações pela web, como imagens, páginas html, arquivos de música, arquivos de vídeo, entre outros. Devido ao fato do HTTP utilizar protocolos de transmissão de dados confiáveis, os dados transmitidos pelo HTTP não irão sofrer danos ou modificações durante o envio, garantindo que o mesmo dado transmitido em uma ponta, chegue à outra sem alterações. A arquitetura do HTTP permite o envio de diversos tipos de dados, sejam eles tipos simples, como arquivos de texto, ou tipos mais complexos, como os arquivos de áudio utilizados no projeto. (GOURLEY e TOTTY, 2002)

O .Net Framework é uma plataforma de desenvolvimento desenvolvida pela Microsoft, com o objetivo de permitir que o código possa ser escrito em diversas linguagens diferentes, porém para um mesmo fim, ou mesmo sistema. O C# é uma linguagem de programação desenvolvida pela Microsoft que trabalha com o .Net Framework e, com o objetivo de causar pouco impacto na migração de outras linguagens, sua sintaxe é similar às linguagens de programação C, C++ e Java, devido ao enorme uso dessas linguagens. (SCHILDT, 2010)

O Android é um sistema operacional desenvolvido principalmente para dispositivos móveis como smartphones e tablets, baseado em uma versão modificada do Linux. Ele também é um sistema gratuito e de código aberto, o que possibilita que outros desenvolvedores modifiquem seu código de acordo com suas necessidades. (LEE, 2011)

1.1 – Motivação

O que gerou a motivação desse projeto foi o desconhecimento, no período em que surgiu a idéia, em meados de 2011, de um dispositivo móvel que, conectado em uma rede local, em ambiente doméstico, fosse capaz de reproduzir as músicas contidas em um computador pessoal sem a necessidade de ter que realizar o armazenamento de forma definitiva dos arquivos de música do computador no dispositivo, sendo que o dispositivo iria reproduzir a música desse computador.

Com a evolução dos smartphones, a ideia do dispositivo foi substituída pela possibilidade de desenvolvimento de um sistema onde usa-se um smartphone, desenvolvendo apenas a parte do software com a devida integração, ao invés de desenvolver todo um dispositivo com hardware e softwares, que seria bem mais complexo e complicado de se aplicar e ainda teria o custo adicional do hardware empregado.

1.2– Objetivo Geral do Trabalho

O objetivo geral é desenvolver um sistema cliente/servidor para transmissão de músicas no formato mp3 do servidor para o cliente através de streaming, onde o cliente será capaz de reproduzir a música.

1.3 – Objetivos Específicos

Para atingir o objetivo geral do trabalho é necessário atingir os seguintes objetivos específicos:

- Com base em pesquisa realizada, definir a melhor forma de realizar a transmissão, isto é, se via RTP ou HTML;
- Estudos das linguagens de programação C# e Java para android, assim como os componentes necessários para implementação do projeto;
- Desenvolvimento da aplicação do servidor em C#;
- Desenvolvimento da aplicação do cliente em Java, integrando a comunicação com o servidor;
- Testes de qualidade e desempenho.

1.4 – Metodologia

A metodologia adotada pelo projeto será, primeiramente, realizar uma pesquisa relacionada à codificação MP3, padrão de redes sem fio, protocolos de rede e os protocolos de aplicação RTP e HTTP, assim como as linguagens de programação necessárias para a realização do projeto.

Após a pesquisa, será realizado um estudo das linguagens de programação foco da implementação, o C# e o Java para android, levantando todos os requisitos para a implementação do cliente e servidor.

Depois de concluído o estudo inicial das linguagens, será dado o início ao desenvolvimento das aplicações cliente e servidor, que será dividida em etapas, de forma a permitir testes intermediários das aplicações, tanto das partes independentes da integração, quanto às partes em que ambos os sistemas devem estar conectados entre si.

Concluindo o desenvolvimento do sistema, serão iniciados os testes de desempenho do sistema, de modo a verificar se o produto resultante atende o objetivo geral do projeto de forma satisfatória.

1.5 – Escopo do Trabalho

Neste trabalho será desenvolvido um sistema cliente/servidor formado por uma aplicação no servidor e outra aplicação no smartphone, como cliente.

A aplicação do servidor será desenvolvida em C# com o Visual Studio 2010, e nela será possível ao usuário configurar suas listas de reprodução e organizar suas músicas nessas listas, sendo que essas músicas configuradas serão as que a aplicação cliente terá disponível para reprodução. Também é a aplicação do servidor que será responsável pelo envio das informações (as listas disponíveis e as músicas da lista solicitada), assim como a transmissão do áudio para o cliente.

A aplicação do cliente será desenvolvida para smartphones android com versão igual ou superior a 4.0 e em Java. Na aplicação cliente, o usuário poderá escolher as listas e as músicas previamente configuradas no servidor, podendo ouvi-las e tendo o controle da reprodução das mesmas.

1.6 – Resultados Esperados

Espera-se na conclusão deste trabalho, o desenvolvimento dessa aplicação, sendo ela capaz de atender os seguintes requisitos:

- Ser possível de criar várias playlists e adicionar músicas, no formato mp3, em cada uma delas através da aplicação do servidor;
- Salvamento automático da configuração das listas de reprodução no servidor, de modo que ao iniciar o programa do servidor, ele seja possível de carregar a última configuração salva;
- Ser possível a comunicação do dispositivo móvel com o servidor, permitindo a busca da lista de playlists e da lista de músicas de uma playlists disponível;
- Ser possível a reprodução da música localizada no servidor através do dispositivo móvel, com a possibilidade de controle da reprodução, assim como pausar, mudar de música e avançar para determinado ponto da música;
- Que o servidor permita a conexão de pelo menos um dispositivo android por vez.

1.7 – Estrutura da Monografia

A monografia está distribuída da seguinte forma:

Capitulo 1 – Introdução: Parte que trata dos objetivos do projeto, sua motivação, qual o seu escopo e os resultados esperados, assim como a monografia está estruturada.

Capitulo 2 – Apresentação do problema: Nesse capítulo são apresentados um detalhamento dos problemas gerais e objetivo do projeto.

Capitulo 3 – Referencial Teórico: Será desenvolvido todo o embasamento teórico necessário para o desenvolvimento da aplicação.

Capitulo 4 – Sistema de Transmissão de áudio: Nessa parte é apresentado o projeto as etapas do projeto e o projeto em si. Também nesse capítulo, serão exibidos os resultados do trabalho desenvolvido.

Capitulo 5 – Resultado final do projeto: Nessa parte serão apresentadas conclusões e sugestões para projetos futuros baseados nesse trabalho.

CAPÍTULO 2 - APRESENTAÇÃO DO PROBLEMA

O problema neste projeto de pesquisa tem como objetivo apresentar um meio prático de se ouvir músicas em aparelhos móveis, sem a necessidade de transferir arquivos de áudio do computador pessoal para o celular, de modo que as músicas fiquem localizadas em apenas um local, no computador, mas seja acessível a outros dispositivos móveis que neste projeto serão smartphones com sistema operacional Android.

Existem diversos sistemas que permitem ao usuário ouvir músicas sem que elas estejam armazenadas no dispositivo de áudio do usuário, como por exemplo rádios on-line, Grooveshark (sistema de gerenciamento similar ao Youtube, porém voltado apenas para músicas), que em alguns casos também possuem sistemas onde é possível acessa-los através de dispositivos móveis, porém nem sempre gratuitos.

O primeiro problema encontrado nesses tipos de serviços é a disponibilidade, pois para o usuário utilizar esses serviços ele precisa estar conectado a internet e o serviço precisa estar disponível, o que não está no controle do usuário. Outro problema é a questão do controle das músicas pelo usuário, tendo em vista que em rádios on-line, elas são exibidas de acordo com as programações das rádios e no caso de outros sistemas, como no caso do Grooveshark, não é possível ter todas as músicas que o usuário deseja, devido a limitações diversas, entre elas problemas relacionados a direitos autorais.

Além dos sistema on-line existe também produtos da Apple capazes de realizar o streaming de media em redes locais, mas não é disponível entre um computador pessoal com sistema operacional *Windows* e um *smartphone android*.

Neste projeto será desenvolvido um sistema cliente/servidor na rede local do usuário, onde seja possível, com as músicas armazenadas em seu computador pessoal (o servidor), criar e administrar diversas listas de reprodução no servidor com suas próprias músicas e reproduzi-las em seu celular, assim como controlar a reprodução delas no celular. No desenvolvimento do sistema serão utilizadas as linguagens de programação C#, no lado do servidor, e Java, para o dispositivo móvel, e para a transmissão do áudio entre as plataformas será utilizado o protocolo de comunicação HTTP (Hiper text transport protocol).

CAPÍTULO 3 – REFERÊNCIAL TEÓRICO PARA RESOLUÇÃO DO PROBLEMA

3.1 – Codificação MPEG-1 de 3 camadas (MP3)

A representação de um sinal digital requer uma frequência de amostragem de 48kHz e uma quantização uniforme de 16 bits por amostragem. Isso resulta em uma taxa de transmissão de 768 bits/s para um som digital monofônico e 2 x 768 bits/s para um som digital estéreo. Devido à frequência de amostragem inferior, normalmente utilizada, de 44,1kHz, a taxa de transmissão de um som digital estéreo de um CD é de 2 x 706bits/s. (MUSMANN, 2003)

Em 1988 a ISO (International Standardization Organization) estabeleceu a MPEG (Motion Picture Expert Group) a missão de criar um padrão de codificação para sinais digitais de áudio e vídeo, a fim de permitir sinais interativos de vídeo e áudio em armazenamento de mídias digitais. O MPEG iniciou com membros de 14 instituições de pesquisa para desenvolver a codificação. Ao final, em 1992, o padrão de codificação MPEG-1, camada 1, 2 e 3 foi desenvolvido e proposto para codificação de sinais de áudio estéreo em 2 x 192 bits/s, 2 x 128 bits/s e 2 x 64 bits/s. Depois, a abreviação “MP3” foi introduzida para substituir o nome extenso do padrão de codificação MPEG-1, camada 3.

O algoritmo padrão de codificação de áudio deveria atender os requisitos do sistema propostos pelo *ISO Proposal Package Description*, tais como:

- Taxa de amostragem: 32, 44.1 e 48Khz;
- Resolução: 16 bits - quantização uniforme;
- Taxa de bit monocanal: 32, 64, 96, 128 e 192 kbps;
- Taxa para estéreo ou bilingue: 128, 192, 256, 384 kbit/s;
- Comprimento de uma unidade endereçavel menor que 1/30 s;
- Atraso total da codificação e decodificação inferior a 80 ms.

Em junho de 1989, 14 algoritmos de codificação foram enviados para atender os requisitos da proposta, que devido as suas semelhanças foram agrupados em quatro grupos de desenvolvimento: codificação por subbandas com mais do que 8 subbandas (MUSICAM),

codificação por transformadas com superposição de blocos (ASPEC), codificação por transformadas sem superposição de blocos (ATAC), codificação por subbandas com menos do que 8 subbandas (SB / ADPCM). Esses 4 modelos foram testados extensamente e, devido a falhas de hardware em dois dos modelos, apenas os modelos MUSICAM e ASPEC puderam ser avaliados completamente.

Em ambos os modelos, os canais esquerdo e direito são de um sinal estéreo de áudio, e são codificados separadamente. A entrada de sinal de um canal é representada em PCM (Pulse Code Modulation) com 48kHz de frequência e 16 bits por amostragem. Primeiramente, o sinal é mapeado no domínio da frequência com uso de transformada (ASPEC) ou através do uso de um banco de filtro de subbandas. (MUSMANN, 2003)

Na segunda etapa, os coeficientes resultantes são normalizados com fatores de escala, que são transmitidos como informação adicional. Na terceira etapa, os coeficientes são quantizados e codificados com um codificador de entropia. A fim de controlar a quantização, os limiares de mascaramento são do erro de quantização, minimizando o erro, são calculados baseados em um modelo psico-acústico. A partir dos limiares de mascaramento, a alocação de bit para os coeficientes é estabelecida. A informação de alocação de bits seleciona um quantizador de uma lista de possíveis quantiadores e a alocação de bits é transmitida como informação adicional. A figura 3.1 apresenta o diagrama de bloco para exemplificar a codificação. (MUSMANN, 2003)

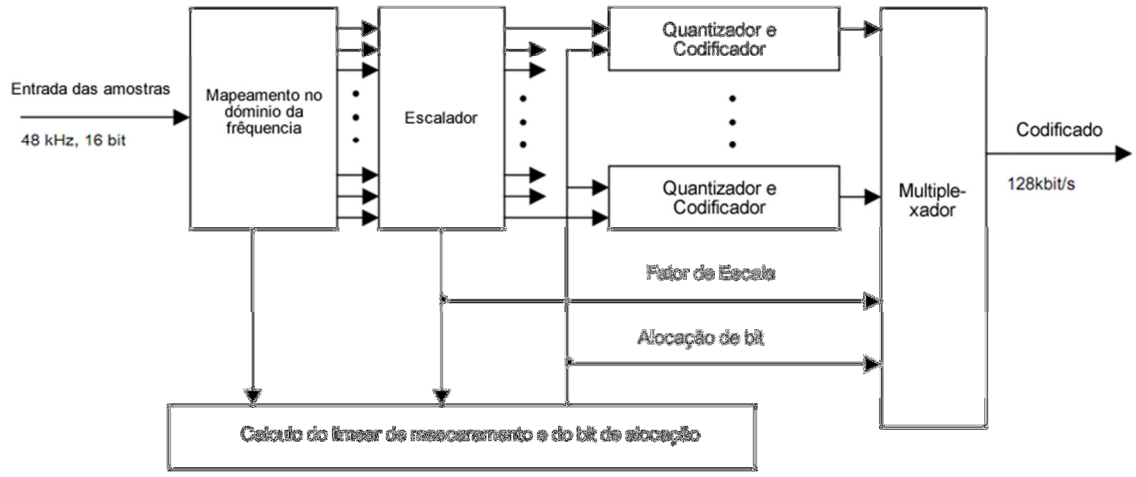

Figura 3. 1 - Diagrama de blocos de um codificador de canal único **(MUSMANN, 2003)**

O padrão de codificação MPEG-1 está dividida em três camadas:

- A camada 1 contém o mapeamento básico da entrada do sinal digital dividida em 32 subbandas, segmentação para formatar os dados em blocos, um modelo psico-acústico para determinar a alocação de bits adaptável e quantização usando blocos extendidos e formatados. A taxta de transmissão recomendada é 2 x 192 bits/s.

- A camada 2 provê uma codificação adicional de alocação de bits usando um fator de escala e amostras. Tamanhos diferentes de *frames* são utilizados. A taxa de transmissão recomendada é de 2 x 128 bits/s.

- A camada 3 apresenta uma melhoria na resolução de frequência baseado em um banco híbrido de filtros. Adiciona um quantizador não uniforme, com segmentação adaptativa e codificação da entropia dos valores quantizados. A taxa recomendada é de 2 x 64 Kbits/s. (MUSMANN, 2003)

Os aspectos abordados neste item apresentam de uma forma resumida os principais passos na evolução e obtenção de um padrão como o MP3, permitindo as codificações em baixas taxas com qualidade, o que permitiu sua rápida popularização e adoção para transmissão de áudio e vídeo pela internet, redes sem fio e redes de celulares. Esse padrão será utilizado no projeto aqui desenvolvido.

3.2 - Redes Wi-Fi IEEE 802.11

A tecnologia Wi-Fi permite a conexão de dispositivos através de redes sem fio. Ela utiliza o padrão IEEE 802.11 que se divide em vários subpadrões, entre eles estão o 802.11b que utiliza as faixas de frequência de 2,4GHz a 2,485GHz e permite uma taxa de dados de até 11 Mbps, a 802.11a que utiliza as faixas de 5,1GHz a 5,8GHz e permite uma taxa de até 54Mbps e a 802.11g e utiliza as faixas de 2,4Ghz a 2,485GHz e permite uma taxa de até 54Mbps.

Os três padrões utilizam muitas características em comum. Todos utilizam o mesmo protocolo de acesso ao meio, a mesma estrutura de quadros na camada de enlace e permitem a utilização dos modos estruturados e ad hoc. (KUROSE e ROSS, 2009)

Uma rede tradicional de dados se constitui de roteadores, conexões ponto-a-ponto e terminais para operação e manutenção. Os elementos de uma rede wireless incluem terminais móveis, pontos de acesso (acess point), roteadores sensíveis à mobilidade, incluindo na estrutura de rede todas as funcionalidades necessárias para mobilidade de rede. (PAHLAVAN e LEVESQUE, 2005)

Uma rede sem fio opera de forma bastante transparente para o usuário, onde cada estação possui um endereço MAC de 6 bytes que é armazenado no suporte lógico inalterável (firmware) do adaptador da estação, assim como também permite o suporte para os protocolos TCP/IP e demais protocolos de rede, de forma similar ao que ocorre nas redes cabeadas, o que permite que um sistema projetado em uma rede sem fio, possa operar basicamente da mesma forma em uma rede cabeada.

O smartphone utilizado neste projeto permite a configuração de ambas os protocolos 802.11b e 802.11g, sendo utilizado o padrão 802.11b por possuir o menor desempenho e com isso, um teste melhor de eficiência poderá ser feito.

3.3 - Streaming

O Streaming, ou fluxo de mídia, é uma forma de distribuir informação de mídia pela internet através de pacotes, com o objetivo de reprodução da mídia, seja áudio ou vídeo, ao mesmo tempo em que a mídia é enviada, sem a necessidade do armazenamento da mídia pelo lado do usuário, apenas em questão de buffer de reprodução, ou seja, o arquivo de mídia é armazenado em memória volátil apenas durante o processo de reprodução, onde ao término da reprodução o arquivo é apagado. (FARIA, 2012)

3.3.1 - Protocolo RTP

O RTP provê serviços necessários para o transporte de media de tempo real, assim como áudio e vídeo, em redes IP. Esses serviços incluem *timing recovery*, detecção e correção de perdas, informações de *payload* e fonte, controle da qualidade de recepção, sincronização de media e controle de usuários. RTP foi originalmente designado para o uso em conferências de multi-sessão, usando um modelo de sessões leves.

O transmissor é responsável por capturar e transformar o dado audiovisual para transmissão, assim como gerar os pacotes RTP. Ele também participa na correção de erros e controle de congestionamento através da adaptação do envio de media em resposta ao *feedback* do receptor. A figura 3.2 mostra um diagrama de blocos do processo de envio: (PERKINS, 2003)

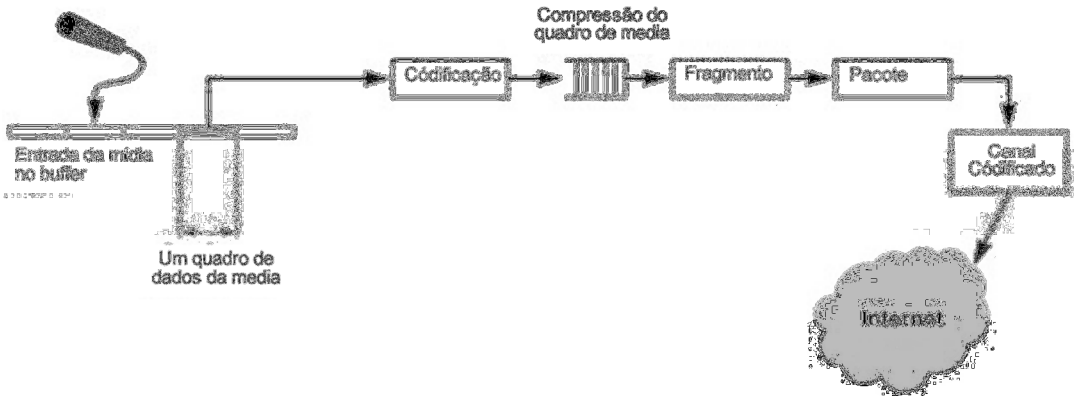

Figura 3. 2 Diagrama de bloco do transmissor RTP **(PERKINS, 2003)**

De acordo com a figura 3.2, inicialmente os dados são capturados e armazenados em *buffer*, onde os quadros comprimidos são gerados. O quadros são codificados de acordo com o algoritmo de compressão utilizado, depois são carregados em pacotes e preparados para o envio. Se os quadros forem grandes, eles são fragmentados em vários pacotes RTP e, se pequenos, podem ser agrupados em um único pacote. Após o envio dos pacotes RTP, o *buffer* de dados é gradativamente liberado.

O receptor é responsável por coletar os pacotes RTP da rede, corrigir qualquer perda, organizar o tempo, descomprimir os dados e apresentar o resultado ao usuário. Ele também deve enviar *feedbacks* de qualidade de recepção ao transmissor, para que o transmissor adapte a transmissão para o receptor e manter a base de sessão de usuários.

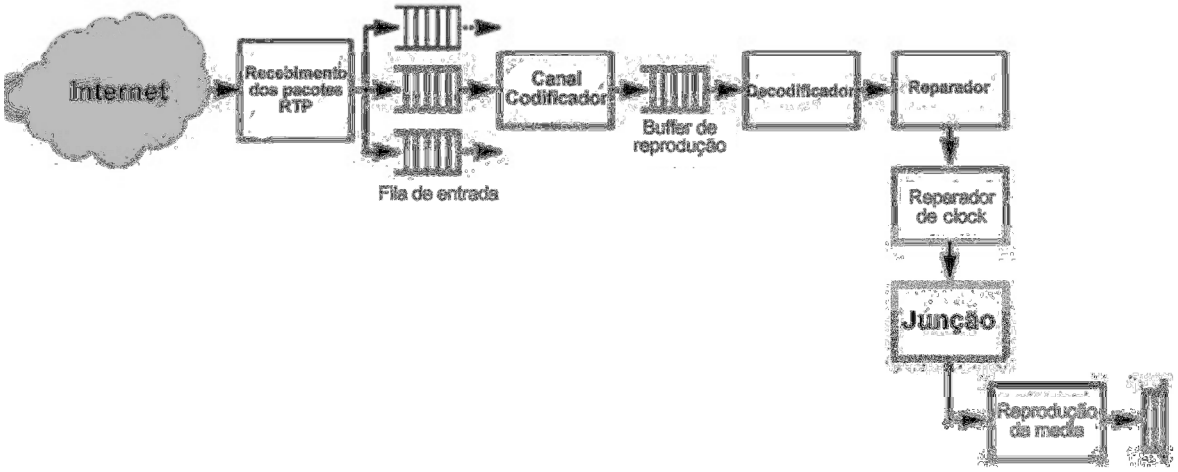

Figura 3. 3 Diagrama de bloco do receptor RTP **(PERKINS, 2003)**

A figura 3.3 mostra um possível diagrama de blocos do receptor RTP, onde primeiramente o receptor coleta os pacotes da rede, valida e os insere em uma fila de entrada. Os pacotes são coletados da fila de entrada e passados por uma rotina opcional de canal de codificação para correção de perdas. Logo em seguida, os pacotes são colocados em buffer de saída. O *buffer* de saída é ordenado pelo *timestamp* e o processo de inserção dos pacotes no buffer corrige os erros de ordenação causados pelo transporte. Os pacotes ficam no buffer até que o quadro seja completamente recebido e eles são armazenados para remover variações internas de pacotes que podem ter sido causadas pela rede.

Quando o tempo de execução é atingido, os pacotes são agrupados para formar quadros completos e os danos ou falta de quadros são reparados. Neste ponto, não pode ser observável diferenças nas taxas de clock nominal do emissor e do receptor. Tais diferenças se manifestam como desvio do valor do relógio media RTP em relação ao relógio de reprodução. O receptor deve compensar essa inclinação do relógio para evitar falhas na reprodução.

O Protocolo RTP está dividido em duas partes: *RTP data transfer protocol* e *RTP control protocol (RTCP).* O RTP *data tranfer protocol* gerencia a entrega dos dados de tempo real, assim como áudio e vídeo, entre os sistemas. Ele define um nível adicional para a execução da media, incorporando uma sequência numérica para detecção de perdas, *timestamp* para permitir a recuperação de temporização, tipos de reprodução e identificação de fonte. O RTCP oferece o feedback de qualidade de recepção, identificação de participantes e a sincronização entre os *streamings* de media. (PERKINS, 2003)

3.3.2 – Protocolo HTTP

O HTTP é um protocolo usado para comunicação no World Wide Web (WWW), seja através de navegadores, servidores ou outras aplicações. O HTTP transfere a maior parte das informações pela web, como imagens, páginas HTML, arquivos de música, arquivos de vídeo, entre outros.

Devido ao fato do HTTP utilizar protocolos de transmissão de dados confiáveis, os dados transmitidos pelo HTTP não irão sofrer danos ou serão alterados durante o envio. Isso é uma vantagem tanto para o usuário, pois não precisa se preocupar com a integridade dos dados, e para o desenvolvedor, que por não se preocupar com a duplicação, distorção ou destruição dos dados enviados, pode se voltar para os detalhes de programação, sem se preocupar tanto com os detalhes da transmissão dos dados. São essas características a que permitem ser utilizado para a aplicação deste projeto, podendo realizar como no caso do RTP a parte da transferência de dados como é feito no RTP data transfer protocol. A diferença é que este último adiciona características (RTCP) de transmissão em tempo real, gerenciamento de entrega, detecção de perdas, recuperação de temporização, entre outros aspectos relacionados à qualidade em ambientes de transmissão de tempo real.

Um servidor HTTP armazena o conteúdo web e provê esses dados quando é requisitado por clientes HTTP. Para definir o tipo do conteúdo que está sendo disponibilizado pelo servidor, ele anexa junto ao conteúdo enviado o MIME type do arquivo enviado. O MIME type é um rótulo textual representado primeiramente pelo tipo básico do conteúdo, seguido por um subtipo, separado por ‘/’, como por exemplo, *audio/mpeg*, onde primeiro nós temos a definição do arquivo sendo um arquivo de áudio, seguindo pelo tipo de codificação do arquivo, no caso o MPEG.

Uma transação HTTP consiste em um comando de requisição, vinda do cliente para o servidor, e uma resposta, do servidor para o cliente. Essa comunicação acontece com blocos formatados de dados chamados mensagens HTTP.

As mensagens HTTP contêm três partes, a primeira parte é uma linha de início, que descreve a mensagem, seguida por um cabeçalho que contém os atributos da mensagem e terminando com um bloco opcional que representa o corpo da mensagem, que no caso de uma resposta do servidor seria o conteúdo em si.

As mensagens estão divididas em dois tipos, mensagens de requisição e mensagens de resposta. Segue abaixo a estrutura de cada um dos tipos de mensagem:

Mensagem de requisição:

<método> <requisição-URL> <versão>

<cabeçalho>

<corpo da mensagem>

Mensagem de resposta

<versão> <status> <frase-de-resposta>

<cabeçalho>

<corpo da mensagem>

A primeira linha em ambas as mensagens é a linha de início, onde toda mensagem deve começar com essa linha que na mensagem de requisição significa o que tem que ser feito e na mensagem de resposta indica o resultado da mensagem. Na mensagem de requisição, a primeira informação é o método de requisição, sendo os métodos mais comuns o GET e o POST. No GET, o cliente apenas requisita uma informação do servidor, não utilizando o corpo da mensagem. No POST, o cliente envia dados utilizando o corpo da mensagem.

A segunda informação da mensagem de requisição é a requisição URL, que é o endereço do recurso que se está solicitando. Em seguida, a versão do HTTP é informada e a linha de início é encerrada com uma quebra de linha.

Na mensagem de reposta, a linha de início segue um padrão um pouco diferente da mensagem de requisição, sendo enviadas as informações de versão, status e frase de resposta.

A versão segue a versão do HTTP em que a mensagem está sendo enviada, logo após tem-se o código de status que indica a situação final da requisição, como por exemplo, o 200 que indica que a requisição foi tratada com sucesso, ou 404 que indica que o recurso solicitado não foi encontrado. A frase de resposta é uma mensagem representando o código de status previamente informado, que serve unicamente para compreensão humana.

No campo de cabeçalho são colocadas informações adicionais da requisição, como por exemplo, o MIME type do arquivo na resposta. Cada linha do cabeçalho representa uma informação diferente e cada uma dessas informações possui um nome e um valor, separados por dois pontos (:). O cabeçalho termina com uma linha em branco e depois dessa linha em branco, as demais informações representam o corpo da mensagem. Segue abaixo um exemplo de mensagem de requisição, utilizando o método POST para exemplificar o uso do corpo da mensagem, e uma mensagem de resposta, exibindo um arquivo de teste. (GOURLEY e TOTTY, 2002)

Mensagem de requisição:

POST /index.html HTTP/1.0

Accept: text/html

Nome=NomePessoa&Idade=99

Mensagem de resposta:

HTTP/1.0 200 OK

Content-Type: text/plain

Mensagem

3.4 – Software

3.4.1 – DotNet C#

O .Net Framework é uma plataforma de desenvolvimento criada pela Microsoft. Ele possui dois componentes principais: o Common Language Runtime (CLR) e o .Net Framework Class Library. O CLR é a base do .Net Framework, responsável por gerenciar o código em tempo de execução, provendo serviços essenciais como gerenciamento de memória, gerenciamento de tarefas e serviços remotos. O *Class Library* é uma coleção de objetos que são utilizadas para o desenvolvimento de aplicativos, desde aplicativos de linha de comando, aplicativos com interface gráfica, até aplicativos em ambiente web.

O .Net possui uma interoperabilidade entre linguagens, que permite que um código escrito em uma linguagem específica interaja com um código escrito em uma linguagem diferente, o que pode maximizar o reuso e melhorar a eficiência no processo de desenvolvimento. O CLR fornece a base necessária para essa interoperabilidade, especificando e forçando um sistema de tipos comum. O sistema de tipo comum define como os tipos são declarados, usados e gerenciados pelo CLR. Isso permite que sejam utilizados vários tipos de linguagem, no caso desse projeto foi utilizado o C#. (http://msdn.microsoft.com/en-us/library/hh425099.aspx, 2012)

O C# foi desenvolvido pela Microsoft no final da década de 90 e sua primeira versão alpha foi lançada em meados de 2000. O C# é diretamente relacionado ao C, C++ e Java, as três principais linguagens mais usadas no mundo. Além disso, no período da criação do C#, muitos desenvolvedores conheciam essas linguagens, o que facilita na migração deles para o C#, não necessitando que o programador tenha que aprender toda a sintaxe dessa nova linguagem, precisando apenas adaptar seus conhecimentos a nova plataforma. (SCHILDT, 2010)

O WCF (Windows Comunication Foundation) é uma biblioteca de desenvolvimento para o desenvolvimento de serviços no dotNet. Através do WCF é possível implementar uma série de padrões de serviços de forma unificada e simplificada, para maiores detalhes acesse: http://msdn.microsoft.com/pt-br/library/vstudio/ms735119(v=vs.90).aspx. No projeto, ele foi utilizado para gerar um WebService para comunicação entre o servidor e o cliente.

3.4.2 Java para android

O Android é um sistema operacional desenvolvido principalmente para dispositivos móveis como smartphones e tablets, baseado em uma versão modificada do Linux. Ele também é um sistema gratuito e de código aberto, o que possibilita que outros desenvolvedores modifiquem seu código de acordo com suas necessidades.

Outra vantagem do sistema operacional é a unificação do desenvolvimento dos aplicativos, com isso, o desenvolvedor se preocupa apenas em desenvolver para o android e o aplicativo desenvolvido deverá ser capaz de funcionar em diversos modelos de aparelhos diferentes, levando em consideração apenas a versão do sistema operacional. (LEE, 2011)

3.4.2.1 Arquitetura do S.O

A arquitetura do android está dividida em cinco seções, kernel, bibliotecas, runtime do android, estrutura de aplicativos e aplicativos:

O kernel é o núcleo do sistema operacional, nele ficam localizados os drivers para comunicação com os componentes de hardware do dispositivo móvel.

As bibliotecas são responsáveis por disponibilizar as principais funcionalidades do sistema, como por exemplo, a biblioteca WebKit que fornece as funcionalidades para navegação na web.

A runtime pertence a mesma camada das bibliotecas e fornece as bibliotecas centrais que permitem aos desenvolvedores escreverem seus aplicativos utilizando a linguagem de programação Java. Nela também se encontra a maquina virtual Dalvik, que permite que cada aplicativo rode em seu próprio processo.

A estrutura de aplicativos funciona como uma camada intermediária entre a camada de aplicativos e a de bibliotecas, provendo APIs (Application Programming Interface) necessárias para o desenvolvimento das aplicações.

A seção de aplicativos é a camada mais alta da arquitetura do android, onde se localizam os aplicativos que são usados pelo usuário, desde os que vem junto com o pacote do sistema operacional inicial, como agenda, despertador, contato, entre outras, até os aplicativos

instalados pelo usuário. A figura 3.4 apresenta um diagrama da organização da arquitetura do android.: (LEE, 2011)

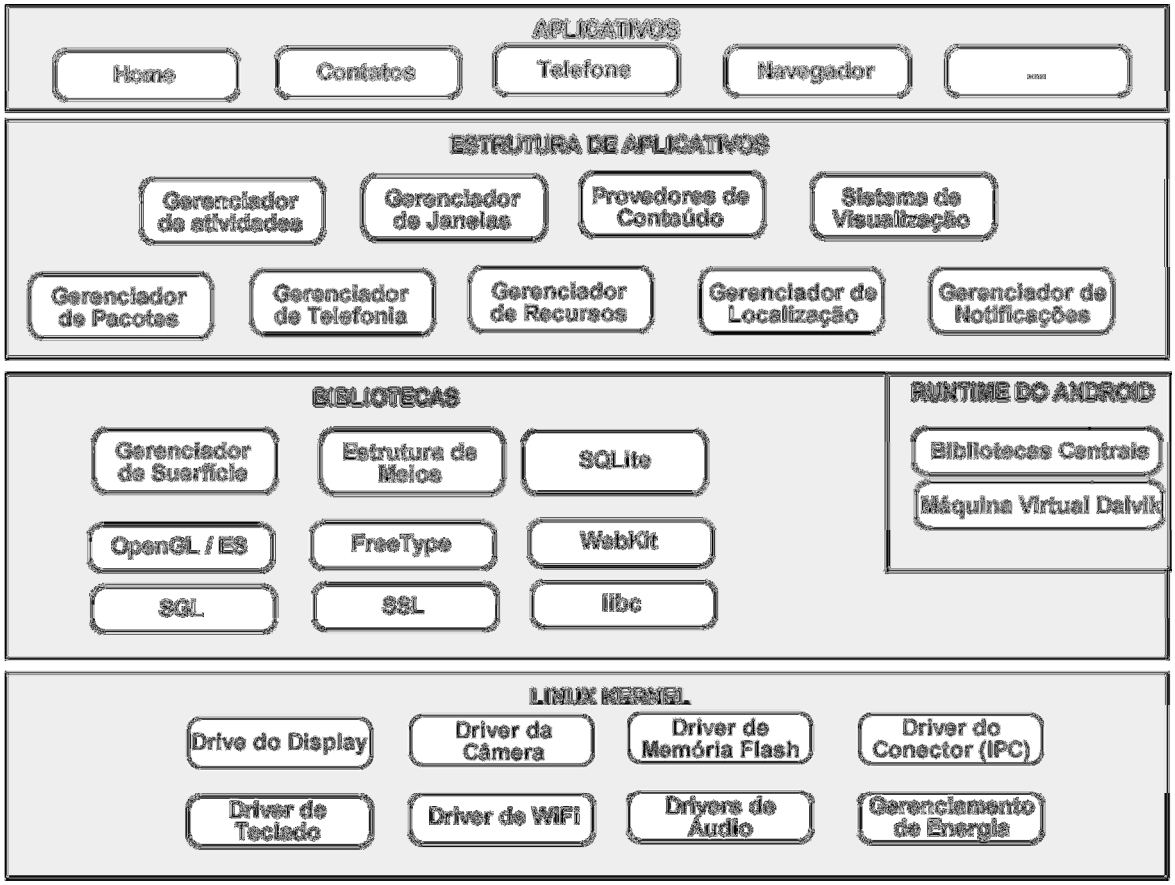

Figura 3.4 Estrutura do Sistema Operacional Android **(LEE, 2011)**

3.4.2.2 – Componentes de desenvolvimento

As aplicações no android podem ser compostas por quatro tipos de componentes: atividades, provedor de conteúdo, serviço e receptores de transações:

Uma atividade representa uma tela de interface com o usuário, ou seja, a interface que é exibida para o usuário, de modo que ele possa visualizar e interagir com o sistema, como por exemplo um reprodutor de música que pode possuir uma atividade para exibir a lista de músicas e outra para exibir a música em reprodução, com os comandos para o usuário interagir com o sistema, mas apesar de estarem em um mesmo aplicativo, cada atividade é independente.

Um serviço é um componente que trabalha em segundo plano para executar operações de um longo período de tempo ou executar um trabalho para um processo remoto. Um exemplo de serviço seria a reprodução de uma música que pode rodar em segundo plano, enquanto o usuário pode utilizar outros aplicativos.

O provedor de conteúdo é responsável por gerenciar e compartilhar os dados do aplicativo. Esses dados podem ser guardados em arquivo no sistema, em uma base de dados do SQLite, na web, ou em qualquer fonte de armazenamento persistente que a aplicação tenha acesso.

Um receptor de transações tem a função de responder a chamadas do sistema, essas chamadas podem tanto vir do sistema, como por exemplo, quando a tela é desligada, ou quando a bateria está fraca, ou vir de outros aplicativos, como ser enviados do aplicativo, como informar a outros aplicativos que uma determinada ação foi concluída ou iniciada. (http://developer.android.com/guide/components/fundamentals.html, 2012)

3.4.2.3 – Atividades

Um aplicativo android usualmente contém uma ou mais atividades e normalmente uma dessas atividades é a atividade *main* do aplicativo, ou seja, é a atividade que é executada quando o aplicativo é iniciado. As atividades estão ligadas entre si e uma atividade pode iniciar outra e sempre que uma atividade nova é iniciada, a anterior é parada, porém ela é salva em pilha para caso o usuário volte a ela.

Cada atividade possui um ciclo de vida, de modo que para cada mudança de estado um evento é disparado e é possível executar ações durante essas mudanças, por exemplo, quando uma atividade é criada, é disparado o evento *onCreate* e assim algumas rotinas de inicialização específicas do aplicativo são rodadas, como criação de variáveis de controle, ou configurações do sistema. Na figura 3.5 é exemplificado um diagrama contendo os ciclos de vida de uma atividade:

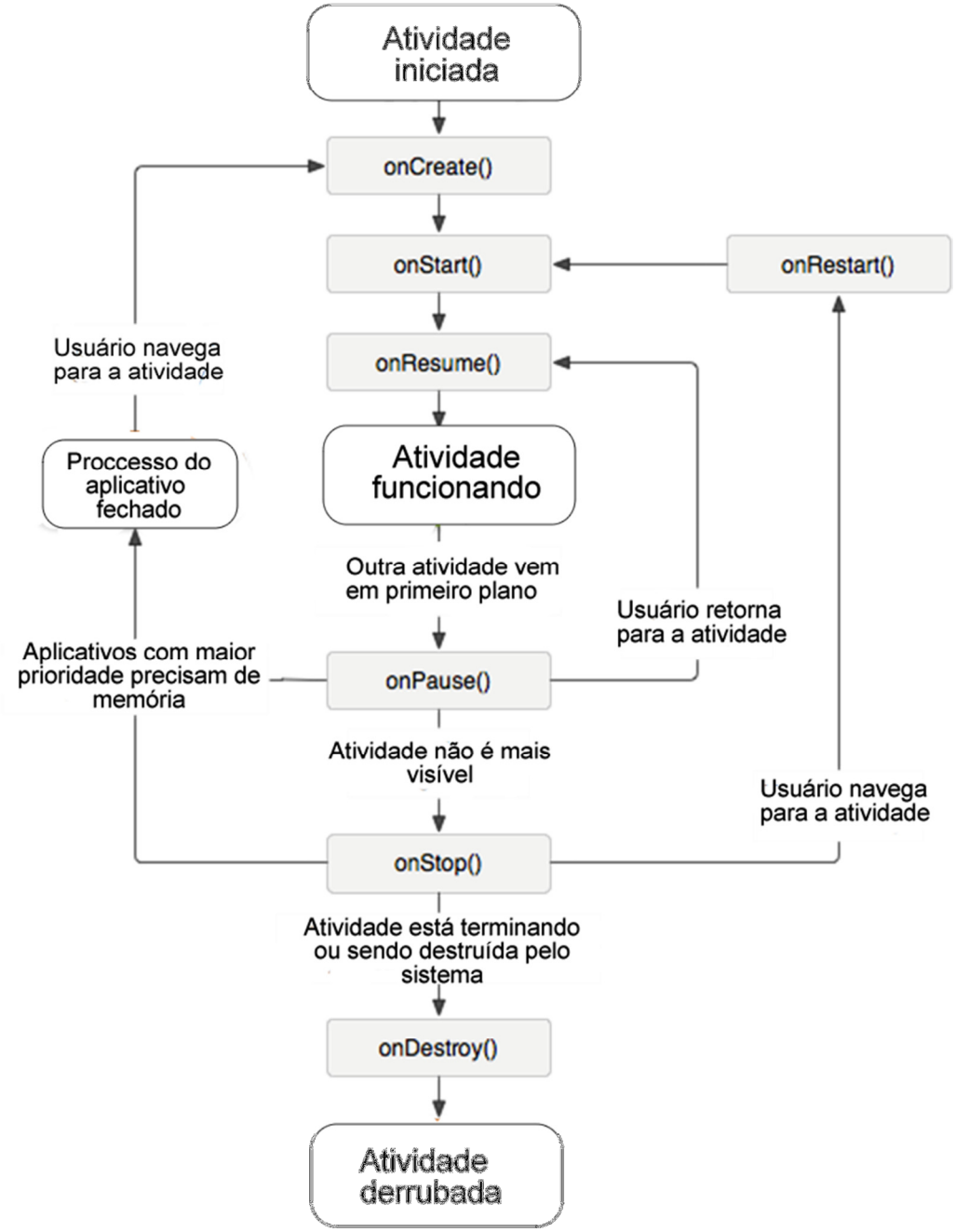

Figura 3. 5 Ciclo de vida de uma atividade

(http://developer.android.com/guide/components/activities.html, 01/11/2012)

Ao iniciar a atividade, o primeiro evento executado é o *onCreate*, sendo ela obrigatória na declaração da classe da atividade, e nela devem ser executadas as rotinas de inicialização da atividade e onde a tela de exibição para o usuário é criada e exibida. Logo após a atividade ficar visível ao usuário, o evento *onStart* é executado. No momento em que se inicia a interação com o usuário é executada o evento onResume, nesse momento a atividade está no topo da pilha de atividades.

No momento em que o sistema está começando a iniciar outra atividade, este método é normalmente utilizado para salvar mudanças de dados, interromper animações e outras ações que consomem o processamento. A atividade que está se sendo iniciada aguarda o fim da execução do evento para poder ser inicializada. O evento *onStop* é executado no momento que a atividade não está mais visível para o usuário, seja por ter sido destruída ou uma atividade é exibida por cima dela. Quando a atividade volta a ficar visível, o evento *onRestart* é executado, logo antes do *onStart*, ou é executada a função *onDestroy* caso a atividade seja destruída. Se, após o processo de *onPause*, a atividade não fique invisível para o usuário, quando ela retorna para a frente da tela, é executado a função onResume novamente. (http://developer.android.com/guide/components/activities.html, 01/11/2012)

No projeto, os conceitos apresentados sobre o C# e o *.Net Framework* são necessários para entendimento e desenvolvimento da aplicação do servidor desktop, utilizando-se o WCF para parte da comunicação entre o servidor e o cliente. Para o desenvolvimento do cliente, será utilizada a programação voltada para android, principalmente na parte de atividades.

CAPÍTULO 4 – SOLUÇÃO PROPOSTA

4.1 – Apresentação Geral da Solução Proposta

O projeto apresentado é um sistema cliente/servidor de streaming de áudio em rede sem fio, onde o servidor é responsável pelo gerenciamento dos arquivos de áudio e o cliente fará a comunicação com o servidor para capturar e reproduzir o áudio gerenciado e organizado pelo servidor.

No servidor é possível a criação de várias listas de reprodução e em cada uma dessas listas pode-se ter várias músicas. Será utilizada a tecnologia WCF no servidor para gerar o *WebService* em SOAP que é responsável pela comunicação com o cliente para transmitir a lista das listas de reprodução disponível e das músicas da lista desejada. A plataforma de desenvolvimento do servidor é o .Net em C#, utilizando a ferramenta de desenvolvimento Visual Studio 2010.

A aplicação cliente irá solicitar as listas de reprodução e de músicas do servidor, utilizando uma biblioteca *opensource* para comunicação com o WebService, e a reprodução do streaming de música utilizando o HTTP.

O protocolo RTP era a primeira opção para a transmissão do streaming de áudio, porém possuía um custo de desenvolvimento elevado para a implementação no sistema, o que inviabilizaria a entrega em tempo hábil do projeto, além de possuir muitas funcionalidades, cuja implementação não era prevista no escopo do projeto e resolução do problema. Na prática, foi implementada parte da funcionalidade da parte 1 do protocolo RTP, no que se refere à transferência de dados, porém, sendo realizada de forma mais simples com o protocolo HTTP. A parte do protocolo HTTP utilizada possui uma grande simplicidade de implementação, o que possibilitou a implementação de streaming com um custo de desenvolvimento reduzido.

A aplicação do servidor foi desenvolvida em .Net C# utilizando a ferramenta de desenvolvimento Visual Studio 2010 e funciona em plataformas Windows. O cliente foi desenvolvido para *smartphones android*, onde foi utilizada a ferramenta eclipse configurada para desenvolvimento para esses tipos de aparelhos.

A figura 4.1 mostra a organização das classes utilizadas no projeto, que está dividida em duas partes, uma no lado do servidor e outra no cliente. No servidor a classe *PlaylistController* é a classe principal do servidor, controlando a inclusão, busca e exclusão

das playlists e músicas, assim como a comunicação com os demais módulos do sistema. A classe *HttpServer* e a interface *Icommunicate* são responsáveis pela comunicação externa entre o servidor e o cliente, recebendo e respondendo as requisições. Como interface para o usuário exitem as classes *Server* e *NovaPlaylist,* onde a primeira é a tela principal para o usuário, onde é possível organizar as playlists do sistema e a segunda é uma tela para inclusão de uma nova playlist no sistema.

No lado do cliente, a classe *Controller* é a classe que controla as interações do sistema, responsável também por realizar a comunicação com o servidor, tanto para busca das playlists e lista de músicas, como o controle da reprodução de músicas. As classes *MainActivity*, *MusicList* e *Player* são responsáveis pela exibição das telas ao usuário, sendo elas para exibir a tela principal, com a lista de *playlists*, a tela que exibe a lista de músicas de uma playlist selecionada e a tela de reprodução da música, respectivamente.

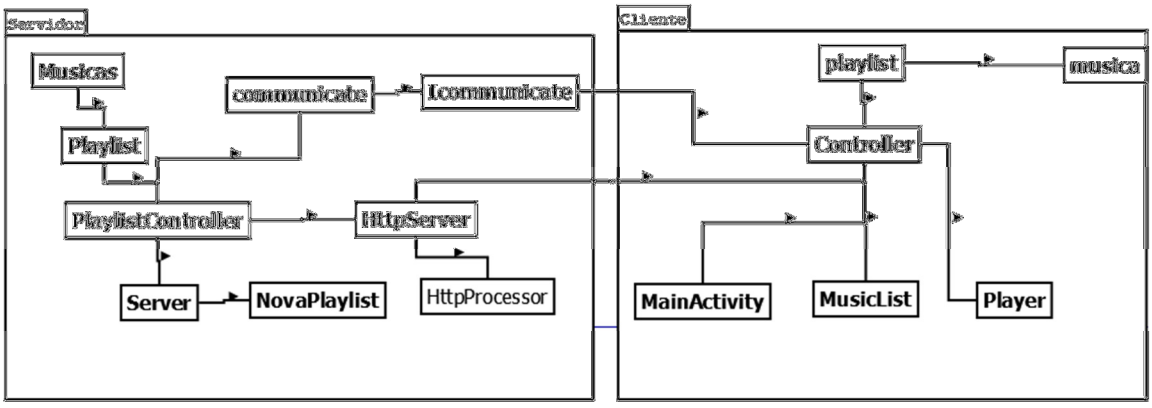

Figura 4. 1 Organização do sistema (Autor)

A figura 4.2 apresenta o fluxograma do servidor, em que inicialmente ele inicializa os serviços do *WebService* e HTTP, ficando disponível para requisições do cliente, logo em seguida fica a espera da interação do usuário, onde ele pode escolher entre adicionar uma nova *playlist*, remover uma *playlist* existente, adicionar uma música a *playlist* selecionada ou remover uma música da *playlist*.

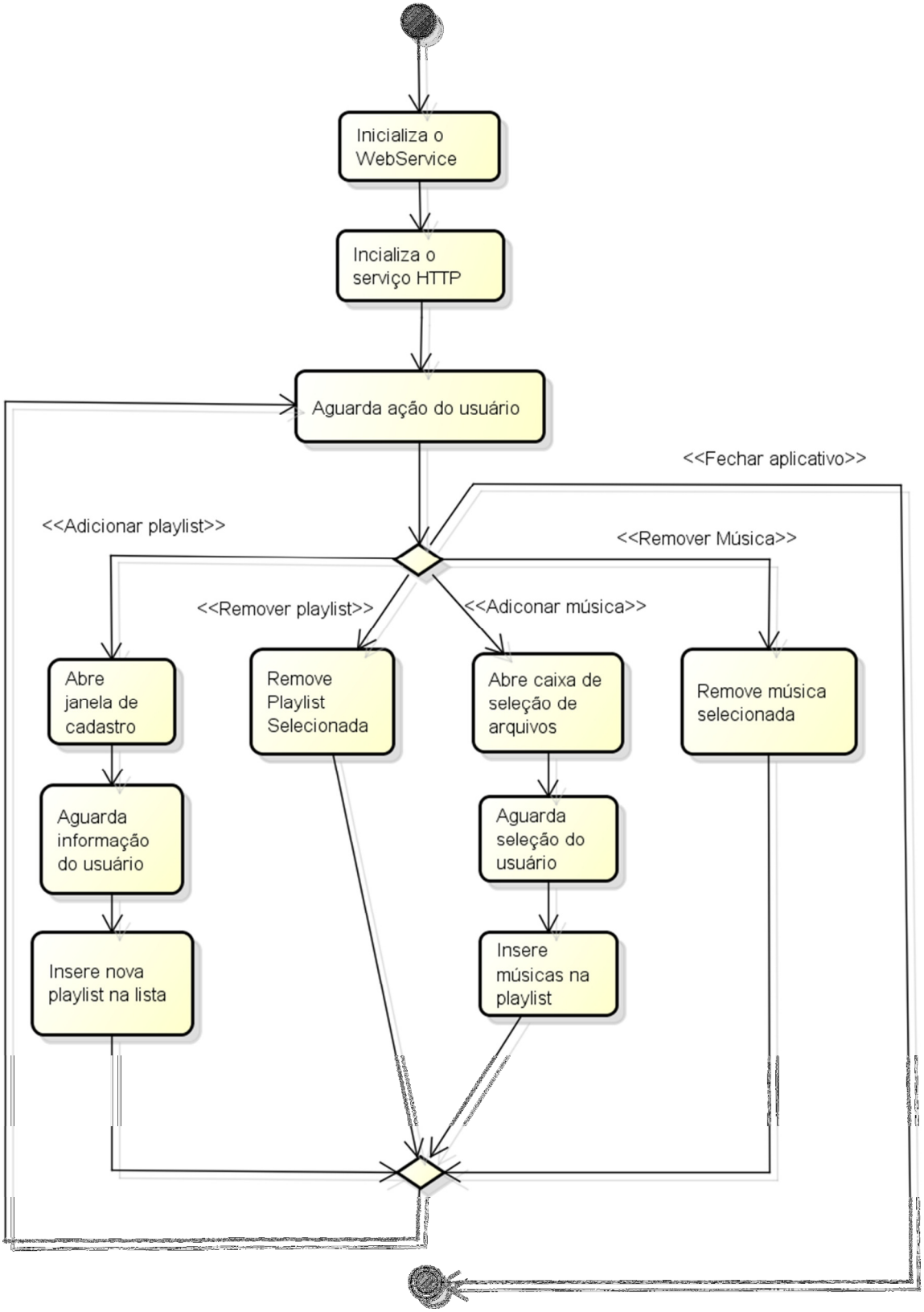

Figura 4. 2 Fluxograma do servidor (Autor)

A figura 4.3 mostra um diagrama representando o fluxo da aplicação cliente com o servidor:

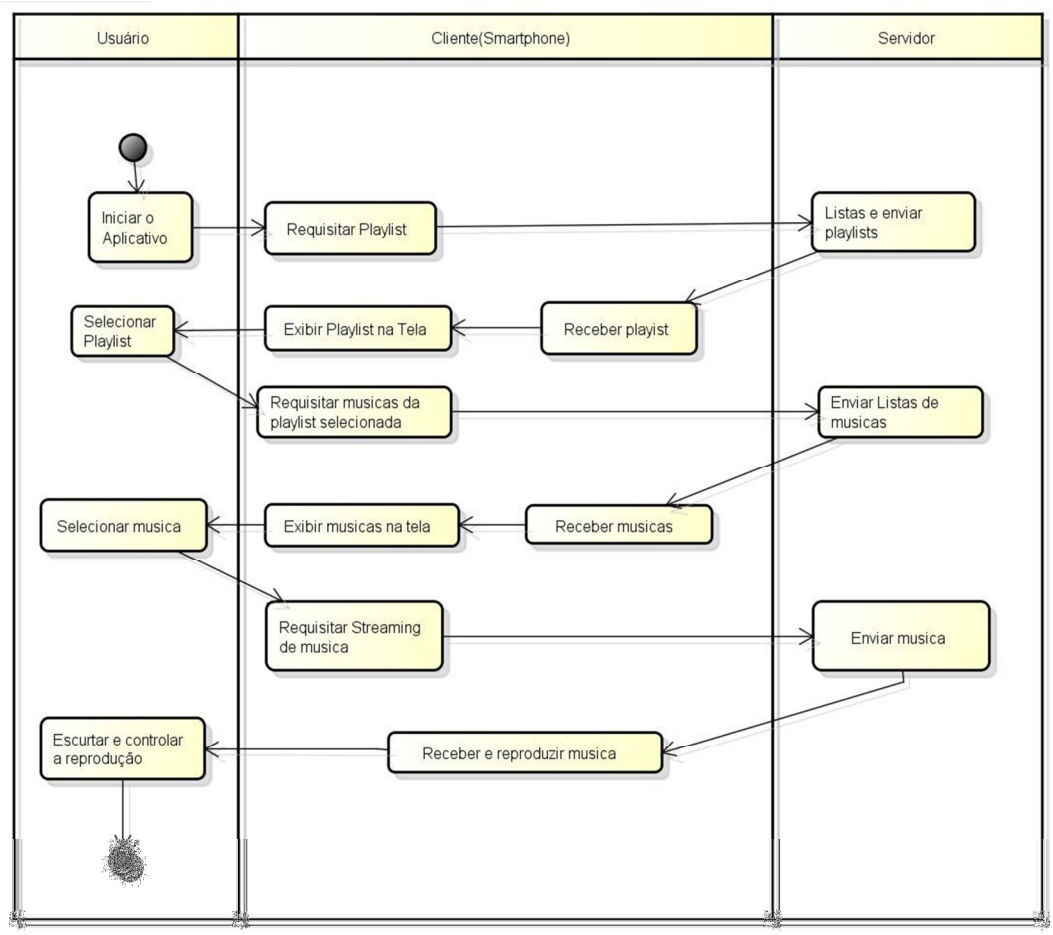

Figura 4. 3 Diagrama de atividade do sistema proposto (Autor)

4.2 – Descrição das Etapas do Projeto

O Desenvolvimento do projeto foi dividido em cinco etapas, desenvolvimento da interface inicial do servidor, interface do aplicativo android, integração inicial entre cliente e servidor, transmissão de streaming de áudio, controles de reprodução e ajustes finais:

Na etapa de desenvolvimento da interface do servidor será criada a interface inicial do sistema e a estrutura inicial do sistema, como a criação e configuração das listas de reprodução, assim como a geração do XML que será usado para salvar as informações do servidor, de modo que sempre que o servidor for iniciado ele carregue essas informações.

Na etapa interface do aplicativo android será criada a parte inicial do android, onde terá a parte de exibição e seleção das listas de reprodução, assim como a exibição das músicas da lista selecionada.

Na etapa inicial entre cliente e servidor será desenvolvido no servidor um serviço utilizando o WCF e SOAP, que permitirá que ao cliente buscar as listas de reprodução configuradas no servidor e as músicas de uma determinada lista desejada. No cliente será criada a parte do sistema que irá se comunicar com o servidor.

Na etapa transmissão de *streaming* de áudio, o *streaming* foi implementado, sendo possível ao cliente a reprodução do streaming de áudio. Nessa etapa, deverá ser criada também a interface inicial de reprodução, porém básica, apenas para exibir a música.

Na etapa controles de reprodução e ajustes finais serão criados os controles de reprodução capazes de controlar a reprodução do streaming de áudio e os ajustes finais na aplicação do cliente serão feitas, como ajustes de layout e tela de configuração, para que seja configurado o endereço do servidor.

4.3 - Descrição da Implementação

4.3.1 – Desenvolvimento da interface inicial do servidor

Na implementação dessa etapa, foi criado um projeto no Visual Studio com o template Windows Form em C#. Nesse projeto foi gerado o *form* inicial e nele foram colocados os elementos *listbox*, para exibir a lista de *playlists* existentes em uma e a lista de músicas da *playlists* selecionada em outra, assim como também os botões de interação com o usuário para incluir e remover as *playlists* e músicas do servidor. Segue na figura 4.4 a tela da interface inicial do servidor. Do lado esquerdo estão os componentes de gerenciamento das *playlists* e do lado direito a lista de músicas.

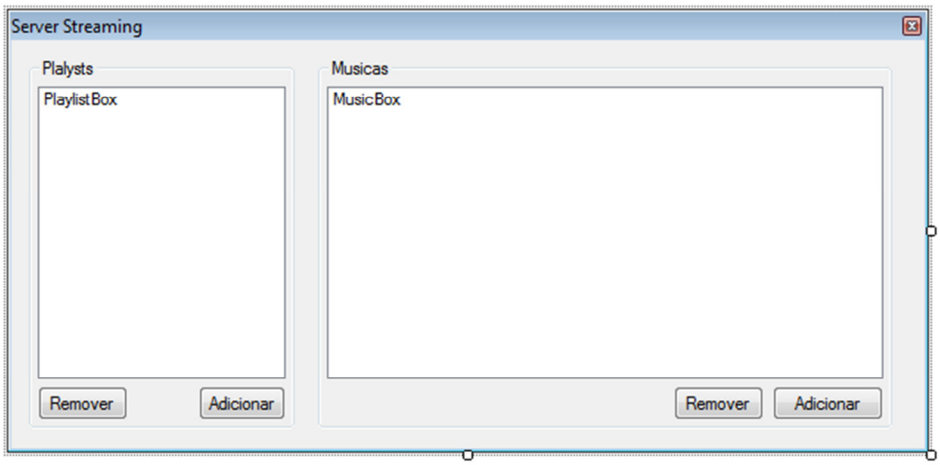

Figura 4. 4 Interface gráfica do servidor (Autor)

Para o botão “Adicionar” do box “*Playlists*” foi incluído um evento para abrir outro Window Form quando for acionado, para que seja informado o nome da nova *playlists* que se deseja inserir. Segue, na figura 4.5, a tela da nova janela e o código responsável por chamá-la:

```
/** Função que abre a janela de criação de nova playlist **/
private void OpenNewPlayList()
{
   // Cria a janela e passa o objeto da janela atual (a principal) para a janela criada
   // Isso é feito para que a janela aberta possa usar as funções da janela principal
   (new NovaPlaylist(this)).ShowDialog();
}
/** Evento onclick do botão "Adicionar" da playlist **/
private void bt_adicionar_playlist_Click(object sender, EventArgs e)
{
   OpenNewPlayList();
}
```

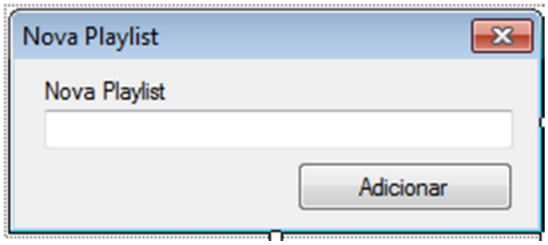

Figura 4. 5 Tela da janela de nova playlists (Autor)

Na janela Nova *Playlist* foi colocado um campo de texto responsável por receber o nome da *playlists* e um botão Adicionar para executar a função que envia a *playlist* para a janela principal. Tanto usando o botão Adicionar, quanto apertando a tecla *enter*, é executada a rotina *AdicionarPlaylist*, que envia à janela principal o nome da *playlist*, onde é verificada se já existe a *playlists* e retornado uma resposta. Ao verificar a resposta da janela principal, a janela Nova *Playlist* é fechada em caso de sucesso, ou emite uma mensagem correspondente a resposta da janela principal. No apêndice A.2 encontra-se o código da implementação.

No botão Remover, na janela principal, o evento onclick executa a função "*rm_playlist_Click*", que por sua vez chama a função "*RemoverPlaylist*" passando a *playlists* selecionada, que executa a remoção *playlist* da lista.

No *listbox* da *playlists*, foi colocado um evento que dispara a função "*PlaylistBox_SelectedIndexChanged*" toda vez que é selecionado um valor diferente na lista e essa função atualiza o *listbox* de músicas para as músicas da *playlists* selecionada.

O botão Adicionar da lista de músicas executa a função bt_adicionar_musica_Click que abre uma caixa de diálogos do tipo *OpenFileDialog* nomeada de *OpenMusicFiles* para que o usuário selecione as músicas que ele deseja adicionar. A caixa de diálogos foi configurada para exibir apenas arquivos de extensão mp3 e para aceitar que sejam selecionados múltiplos arquivos. O botão "Remover" da lista de músicas executa a função "*bt_remover_musica_Click*" no evento click, que remove a música da *playlist*. Encontra-se no apêndice A.1 a implementação das funções responsáveis pela gestão de músicas. A figura 4.6 mostra a configuração do componente *OpenFileDialog* usada:

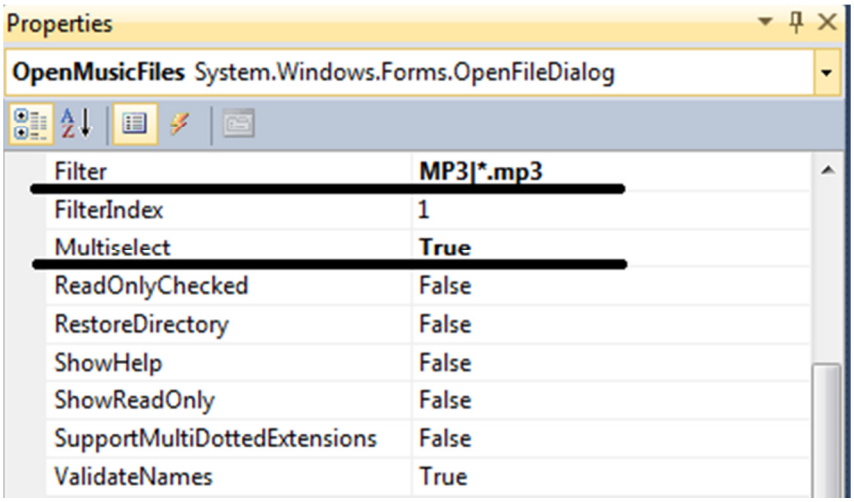

Figura 4. 6 Configuração da caixa de dialogo de seleção de músicas (Autor)

Para o controle das *playlists* e músicas foram criados três classes: *Musicas*, *Playlist* e *PlaylistController*:

A classe *Musicas* guarda as informações da música, como o path onde o arquivo está salvo, nome do arquivo, titulo da música, artista, álbum e o ID da música, gerado através de

uma variável de controle chamada last_id, que é estática e incrementada toda vez que é gerado uma música nova. Caso o ID seja enviado na criação do objeto, o valor enviado é assumido e se o last_id for menor que o valor enviado na criação, o *last_id* recebe o valor enviado incrementado mais um. A classe usa uma biblioteca externa chamada ID3, que é responsável por buscar as informações do arquivo de música do arquivo e se encontra no site *CodeProject* no artigo Do Anything With ID3, referenciada na bibliografia (J.I, 2007). A classe implementada encontra-se no apêndice A.6.

A classe *Playlist* guarda as informações da *playlist*, como nome da playlists, ID, gerado do mesmo modo que na classe de músicas. Ela também é responsável por gerenciar sua lista de músicas, assim com salvar e carregar a lista de músicas em arquivo xml. Através da classe é possível buscar uma música por seu ID, verificar a quantidade de músicas existentes, adicionar novas músicas, controlando para que não sejam inseridas músicas repetidas, gerar e salvar o XML da lista de músicas, que é realizada sempre que um item é incluído ou removido da lista, e também apagar o xml gerado. O código implementado da classe encontra-se no apêndice A.4.

A classe *PlaylistController* é responsável por manter e gerenciar a lista de *playlists*, assim como ser a classe de comunicação com as demais partes do sistema. É através dela que as janelas e os sistemas de comunicação com dispositivos externos salvam e recuperam as informações salvas no servidor. A classe foi feita de forma que só é possível ter um único objeto instanciado na aplicação e todos os objetos criados em outras classes recebem a referencia do mesmo objeto. No apêndice A.3 é apresentado o código da implementação.

A última parte da implementação dessa etapa é a inicialização do *controler* no sistema, que inicializa a instância e lista as *playlists* do sistema, que foram carregados do XML de playlists. O método *InicialiarSistema* foi implementado na classe *Server*, e é mostrado a seguir:

```
/** Função que Inicializa os serviços do sistema **/
public void InicialiarSistema()       {
   // Pega a instância do controller
   controler = PlaylistController.getInstance();
   // Adiciona a lista de playlists no listbox da playlist
   foreach (var playlist in controler.playlists)          {
      PlaylistBox.Items.Add(playlist.name);   }
}
```

4.3.2 – Interface do aplicativo android

Na implementação dessa etapa foi gerado um projeto de desenvolvimento android com Eclipse, criada as telas de exibição de *playlists* e músicas, sem a integração com o servidor que é o terceiro item do desenvolvimento, isto é, o terceiro item de desenvolvimento é responsável por tratar a comunicação do cliente com o servidor para troca simples de informações, como a pesquisa das *playlists* e das músicas da *playlists* selecionada.

Em ambas as telas de exibição, tanto de *playlists* quando de músicas, foram colocadas no XML de *layout*, um *listview*, para exibir a lista de itens, um *progressbar*, para exibir uma imagem de carregando enquanto as informações são carregadas, e um campo de texto para exibir as mensagens de erro. Cada um desses componentes possuem seus próprios ids para serem identificados pelo sistema. Nos apêndices B.2 e B.4 estão os XMLs da tela de *playlists* e a tela de músicas do aplicativo.

A *activity* que exibe as listas de reprodução é a activity principal do sistema, executada logo na inicialização. Nessa *activity* foram criadas funções de controle para carregar a playlists, selecionar e exibir as mensagens de erro. Essas implementações encontram-se no apêndice B.1.

Funções similares às criadas para a tela das playlists foram criadas para a tela de músicas, com a diferença que nesse momento nenhuma ação foi configurada para quando alguma música é selecionada. No apêndice B.3 está o código implementado.

Assim como na etapa da interface do servidor, na aplicação do servidor android foram criadas três classes de controle: *musica*, *playlist* e *controller*:

A classe *musica* apenas salva as informações da música, como título, artista, id e a descrição que é exibida. Todas essas informações são carregadas do servidor. O código da implementação encontra-se no apêndice B.7.

A classe *playlists* é responsável por armazenas as músicas, nome e id da playlists, seu código encontra-se no apêndice B.6.

A classe *controller* tem mesma função do PlaylistController, que é concentrar a manipulação dos dados em uma única classe unificada. É através dela que os dados do servidor serão buscados e é nela que as informações de *playlists* e músicas ficam guardadas. Nessa etapa foram implementadas as funções que selecionam e retornam as *playlists* selecionadas e suas músicas e duas funções que são implementadas na próxima etapa, *readPlaylist* e *readMusicPlaylist*, responsáveis por buscar informações no servidor das *playlists* e músicas. No apêndice B.5 é apresentado o código da classe controller.

4.3.3 – Integração inicial entre cliente e servidor

No servidor foi criado um WebService utilizando o WCF da Microsoft. Inicialmente foi criado a classe de comunicação com o cliente, um *service contract*, e os objetos *data contract*, que são retornados para o cliente pelo servidor. Foram desenvolvidos quatro *data contract* para a comunicação com o servidor: *cplaylist, playlistResponse, cmusicas e musicaResponse*.

Os *data contract cplaylist* e *cmusicas* são usados para encapsular as informações das *playlists* e das músicas e são membros dos *data contract playlistResponse* e *musicaResponse*, respectivamente. Por sua vez os *data contract playlistResponse* e *musicaResponse* são os objetos que serão enviados para o cliente e ambos contem uma variável de controle que indica se houve algum erro na busca das informações e a lista das informações.

O *service contract* communicate é o responsável pelas operações de comunicação entre cliente e servidor e através dele é possível buscas as informações de músicas e *playlists*. Seu código está dividido em duas partes, a primeira é a interface que define as funções que fazem parte do *WebService* e a classe que herda a interface e onde é implementado a funcionalidade. Foram implementadas duas funções, *getPlaylist* e *getMusica*, responsáveis por retornar as listas de playlists e músicas respectivamente. No apêndice A.6 encontram-se as classes de *data contract* e a interface *Icomunicate* e no apêndice A.7 encontra-se a classe *communicate* implementada.

Para configuração do servidor foi criado um arquivo na raiz da aplicação chamado app.config, que salva as configurações do serviço, assim como a porta de comunicação e qual o *service contract* estará disponível nessa porta. O XML de configuração encontra-se no apêndice A.8.

Para inicializar o WebService, a função *InicialiarSistema*, da janela principal do sistema, foi alterada e foi criado um objeto *ServiceHost* para controlar o serviço, a inicialização do serviço foi adicionada no método *InicialiarSistema* da classe *Server* mostrada no apêndice A.1, segue abaixo o código acrescentado na função:

```
ServiceHost host;
// Inicializa serviço WEB Service WCF
host = new ServiceHost(typeof(communicate.communicate), new Uri[] { });
host.Open();
```

Na programação do aplicativo android, foi utilizada uma biblioteca chamada *ksoap2*, que possui métodos para comunicação com o *WebService*, onde através dele, foi possível executar os métodos do webservice do servidor e fechar a comunicação entre eles. Foram criados dois métodos na classe *controller*, disponível no apêndice B.6. O primeiro método é o *readPlaylist*, responsável por buscar as listas de *playlists* disponíveis e o segundo é o *readMusicPlaylist*, que pega a *playlists* selecionada no dispositivo e busca as músicas para essa *playlists*.

Ambos os métodos funcionam de forma similar, inicialmente é criado um objeto *SoapObject*, responsável por configurar o método do servidor a ser acessado e os parâmetros que serão transmitidos, logo em seguida um objeto *SoapSerializationEnvelope*, responsável por encapsular e preparar o pacote da requisição. Em seguida, com o objeto de transporte, que é responsável por enviar o objeto de requisição serializado e recuperar a resposta. Após a criação dos objetos, utilizando a chamada *call* do objeto de transporte, é passado o objeto serializado com a requisição da mensagem e, com isso, a resposta é tratada e salva nos objetos finais, como playlists ou música. Em caso de falha, a flag status_consulta é setada com o valor -1, para que a tela do android exiba uma mensagem de erro.

4.3.4 – Transmissão de streaming de áudio

Para a transmissão do áudio do servidor, foi criado um serviço HTTP simplificado que tem a função de receber a requisição, processar e enviar a música solicitada. Esse processo é executado por duas classes, *HttpServer* e *HttpProcessor*, que foram adaptadas de um código existente em http://www.codeproject.com/Articles/137979/Simple-HTTP-Server-in-C. O código implementado dessas duas classes encontram-se nos apêndices A.9 e A.10.

Inicialmente o objeto *HttpServer* é criado e inicializado, no método *InicialiarSistema*, na classe Server, passando a porta de conexão do serviço, logo em seguida, é criado uma *thread* para controlar a chegada de requisições pelo método *listen* do objeto criado. A partir desse ponto, o objeto *HttpServer* controla a porta indicada e, para cada requisição efetuada, cria um objeto *HttpProcessor* e dispara uma *thread* com o método *process* do objeto criado.

O método *process* é responsável por tratar a requisição e o retorno da requisição. Na primeira parte, ele instancia um *buffer* com a entrada da requisição e dispara os métodos *parseRequest* e *handleRequest* em sequencia. O método *parseRequest* é responsável por pegar a URL solicitada pelo usuário e o *handleRequest* é um método do objeto *HttpServer* e realiza o processo de buscar a música solicitada e enviar o cliente solicitante. No processo do *handleRequest*, ele pega a URL da requisição e, através dela, pega o id da playlist e da música

e busca a música no controller. Caso a música não seja encontrada, é disparada uma exceção que envia uma mensagem de erro para o cliente, e em caso de sucesso a música e lida, o cabeçalho de sucesso é criado e a música é enviada para o cliente.

No cliente android, foram criadas dois métodos na classe *controller*, *IniciarReproducao* e *encerrarMediaPlayer*. O primeiro é executado quando uma música é selecionada, ele reinicializa o objeto *player*, responsável pela reprodução, seta a URL do servidor, com os ids da playlists e música, e manda preparar a execução com o método prepareAsync, após isso, é criado um evento *setOnPreparedListener*, que será disparado no momento em que a reprodução puder ser iniciada, ou seja, após ter iniciado o download da música e tiver com uma parte mínima do arquivo de áudio para ser reproduzido. A implementação desse código se encontra na classe controller, no apêndice B.5.

4.3.5 – Controles de reprodução e ajustes finais

Nessa etapa final de desenvolvimento foram criadas as funções de reprodução de música, com os controles de reprodução e uma tela de configuração para que o endereço IP do servidor fosse configurado e salvo para que não seja necessário configurar o endereço sempre ao abrir o aplicativo. Para facilitar a configuração do IP no aplicativo, foi colocado na tela do servidor um campo de texto que contém o endereço de rede que deve ser configurado.

Na criação da tela de reprodução e os controles de reprodução foi criado uma nova atividade e nela criados os campos que mostram qual é o artista e a música, o tempo de duração total, a posição atual da música, uma barra de reprodução e os botões de avançar a pausar e voltar. Para todos esses elementos foi atribuído um ID para a manipulação e alteração de informação. Para o carregamento dessas informações foi criado na atividade a função *AtualizarTela*, responsável por buscar as informações da música e preencher na tela, assim como inicializar as variáveis de controle, assim como configurar os eventos de ação dos botões e da barra de reprodução. A função *updateBuffer* é responsável por exibir na tela a situação atual do buffer do streaming de música. A função *startSeek* é responsável por atualizar a situação da reprodução da música, mostrando o tempo atual e movendo a barra de reprodução de acordo com a posição da música. A implementação do código descrito acima, assim como o XML do layout criado, encontram-se no apêndice B.8 e B.9. A figura 4.7 mostra a tela de reprodução em funcionamento.

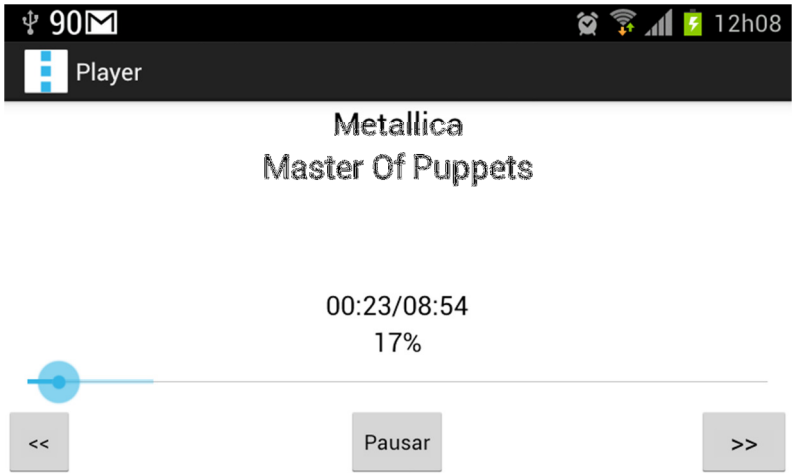

Figura 4. 7 Tela de reprodução da música (Autor)

Para o controle do player foi inserido na classe *controller* as funções e variáveis de controle do *player*, assim como atualizada a função de iniciar reprodução, criada na etapa de transmissão de *streaming*, para adicionar os controles adicionais da tela. Foi criado uma variável no *controler* para receber a referencia da classe da tela, necessária para o controle enviar comandos para ela.

Na função que inicia a reprodução, foram inseridos no objeto do *player*, o evento para avançar a música quando ela acabar e o evento de quando o *buffer* é atualizado. As funções *ProximaMusica* e *VoltarMusica* são responsáveis pelo controle de qual será a próxima música a ser reproduzida, fazendo o controle também de voltar para a primeira música quando não houver próxima música, ou tocar a última música caso o voltar for acionado na primeira música. Em todas as execuções acima, o controler dispara um função da tela para atualizar a exibição para o usuário. Por ultimo foram criadas as funções *PausarVoltar* e *PlayerGoTo*, a primeira é responsável por trocar o status de reprodução da música entre tocando e pausado e a segunda para avançar a música para determinado ponto solicitado. Essas alterações foram feitas no arquivo exibido no apêndice B.5.

Para a configuração do sistema foi criada uma nova tela, onde é possível o usuário colocar o endereço do servidor. Nessa tela foram colocados um campo de texto e um botão para salvar as informações, no *controller* foram criadas as funções *getConfiguracao* e *salvarURL*, para buscar e salvar as configurações utilizando a classe *SharedPreferences*, nativa do android, que é responsável por salvar e recuperar as informações salvas pelo sistema. Para utilização dessa classe, foi necessário criar uma variável no *controller* que recebe um objeto com as funções necessárias para acessar e salvar as informações no sistema. A implementação do código da tela de configuração e o XML do layout encontram-se no apêndice B.10 e B.11. A figura 4.8 mostra a tela de configurações em funcionamento.

Figura 4. 8 Tela de configuração (Autor)

Para facilitar na localização do endereço IP do servidor, a tela principal do servidor foi alterada para buscar e exibir seu endereço IP. A figura 4.9 mostra a tela do servidor alterada.

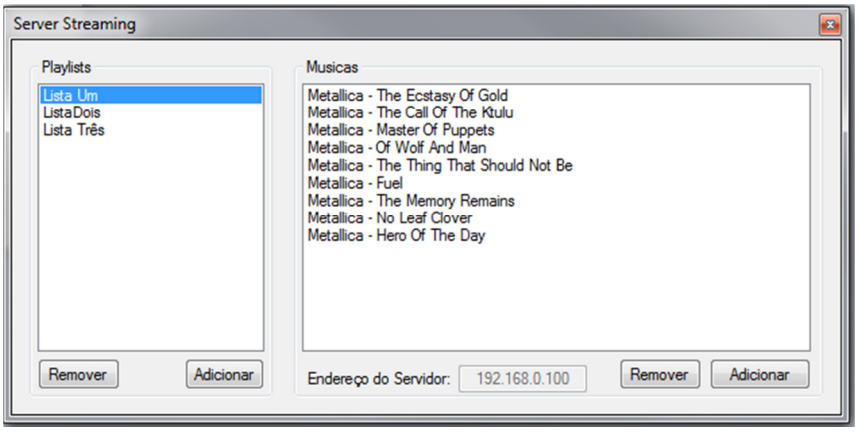

Figura 4. 9 Tela do servidor com endereço IP (Autor)

4.4 - Aplicação Prática do Modelo Proposto

4.4.1 - Apresentação da área de Aplicação do modelo

O sistema apresentado neste projeto foi desenvolvido com o intuito inicial de servir como um sistema doméstico, onde o usuário poderia utiliza-lo para reduzir os trabalhos realizados para transferir a música do seu computador principal, onde toda a sua biblioteca de músicas está armazenada, para o celular, que normalmente possui uma capacidade de armazenamento inferior, o que faria com que o usuário devesse sempre remover as músicas e

colocar músicas novas. Com esse modelo, as músicas estariam sempre no computador pessoal e o celular iria reproduzi-lo diretamente do servidor, sem a necessidade de troca de arquivos e sem se preocupar com a capacidade de armazenamento do celular, já que a música só ficaria no celular por tempo suficiente para a sua reprodução. Assim, a reprodução da música ocorreria utilizando os recurso do celular.

O sistema foi projetado para ser utilizado com um servidor, com o sistema operacional Windows, e um ou mais clientes, com o sistema operacional android e em redes locais sem fio, porém a arquitetura projetada do servidor não impede que outros tipos de clientes sejam criados e que eles possam utilizar de forma igual ou similar os recursos disponíveis do servidor. Ele também não fica limitado apenas as redes locais sem fio, podendo, dependendo da configuração de rede ou internet, ser acessado de forma remota, devido a utilização da arquitetura padrão de rede.

4.4.2 – Descrição da Aplicação do Modelo

Para a utilização do sistema, primeiramente é necessário que o computador pessoal, que será o servidor das músicas, e o smartphone android estejam conectados em uma mesma rede e que as portas de rede do servidor 9000 e 9001, as portas utilizadas pelo servidor para a comunicação, estejam liberadas.

Com isso, antes de utilizar o sistema, é necessário que as músicas sejam organizadas no servidor, criando-se as listas e em seguida inserindo as músicas em cada lista. Quando o servidor é iniciado, ele já carrega as configurações e fica preparado para as comunicações do cliente. A figura 4.10 mostra a tela do servidor com as listas e músicas organizadas, do lado esquerdo estão localizadas as *playlists* e do lado direito as músicas da *playlists* selecionada, abaixo da lista de músicas está localizado o endereço IP do servidor para ser configurado no cliente.

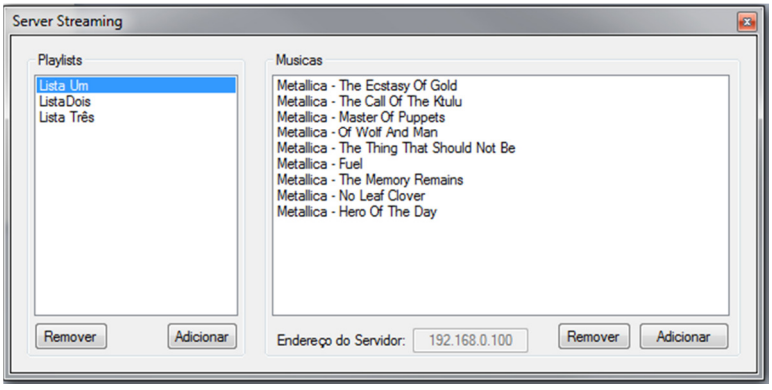

Figura 4. 10 Tela do servidor em uso (Autor)

Com essas configurações o cliente pode ser iniciado e configurado para a utilização e para isso é necessário configurar o endereço do servidor no aplicativo conforme a figura 4.11.

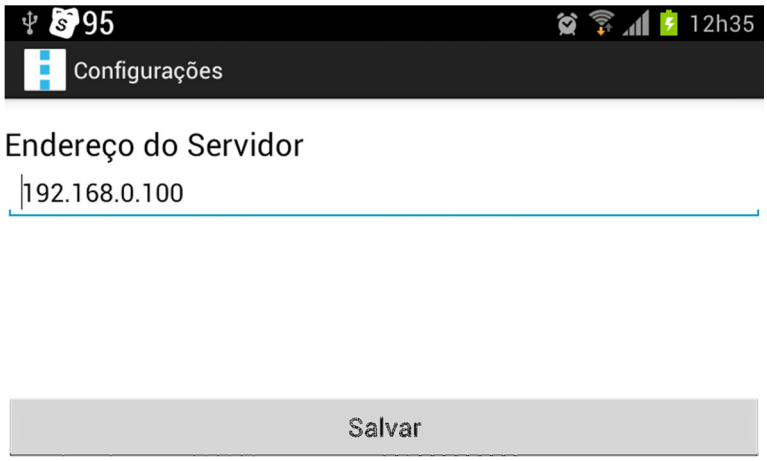

Figura 4. 11 Tela de configuração do servidor (Autor)

Sempre que o aplicativo é iniciado, ele tenta buscar as *playlists* no endereço configurado e caso o aplicativo não consiga conectar ao servidor, ele exibirá a mensagem "Não foi possível conectar ao servidor", como mostrado na figura 4.12.

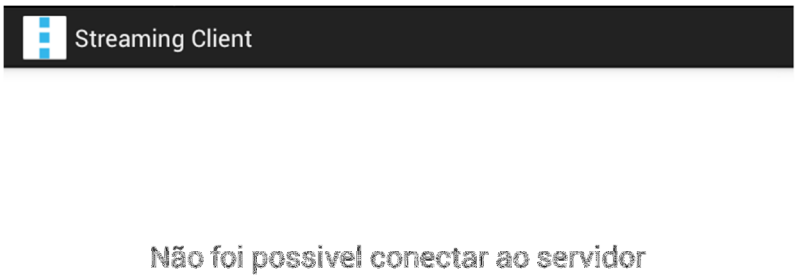

Figura 4. 12 Mensagem de servidor não encontrado (Autor)

Para acessar a tela de configurações ou atualizar as informações do servidor novamente, existe um menu do sistema, acessível tanto da lista de *playlists* quanto da lista de músicas, como apresentado na tela da figura 4.13, que também apresenta a lista de playlists.

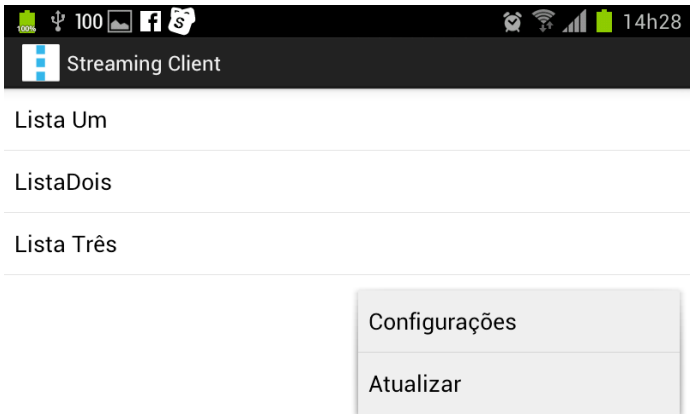

Figura 4. 13 Menu de configuração (Autor)

Ao selecionar uma *playlists*, o cliente irá buscar as músicas no servidor e exibi-las na tela para o usuário, conforme ilustra a figura 4.14.

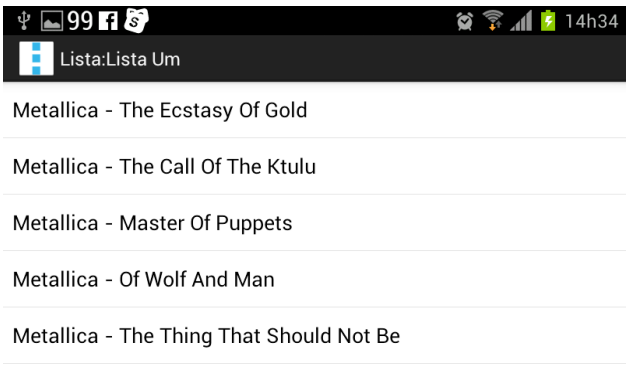

Figura 4. 14 Lista de músicas (Autor)

Ao selecionar uma música, o sistema vai para a tela de reprodução, onde as músicas serão exibidas e tocadas pelo sistema. A barra de reprodução e o tempo de música são atualizadas a cada segundo, assim com também são atualizadas as informações de carregamento da música no buffer, conforme visto na figura 4.15.

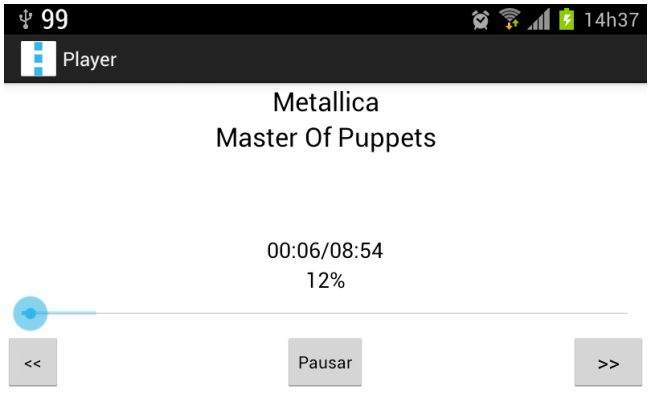

Figura 4. 15 Tela de reprodução de música (Autor)

4.4.3 – Resultados da Aplicação do Modelo

Para testes da aplicação, foram usadas nove músicas de tamanho diferentes em três ambientes diferentes, O primeiro é o mais comum, utilizando a estrutura padrão da rede, com um computador pessoal, um roteador, e um smartphone. No segundo ambiente, o celular é utilizado como o próprio roteador, onde se espera uma conexão e transferências mais rápidas, pois é eliminado um ponto da rede e a transferência sendo feita de forma direta. No terceiro ambiente, é utilizada uma funcionalidade do celular que permite a criação de uma rede através do cabo de dados, nesse passo, apesar de não ser usual, pois necessita que o celular esteja próximo ao computador, é esperado o melhor resultado, devido ao fato da conexão entre os dois estarem sendo feitas direta.

Para uma melhor assertividade dos resultados, cada música será reproduzida três vezes em cada ambiente, e será utilizada a média para análise de resultados. Serão observados dois fatores durante a reprodução das músicas, o primeiro é o tempo de início da reprodução, basicamente o intervalo entre a requisição da música e quando a ela começa a ser tocada, o segundo fator é o tempo em que o sistema termina efetivamente de baixar toda a música, colocando a em um buffer de reprodução.

Na tabela 1 as músicas utilizadas para os testes, assim como o tamanho e a duração de cada uma delas.

Tabela 1 Músicas utilizadas para teste

Número	Música	Tamanho (MB)	Duração (mm:ss)
1	Life in a Day Soundtrack - Piano Theme.mp3	1,269	01:21
2	Jamie Cullum & Götz Alsmann - Georgia on my Mind.mp3	1,996	02:07
3	04 - Prelude.mp3	2,267	00:57
4	La Mer. Charles Trénet.mp3	3,178	03:23
5	Isbells - Maybe.mp3	9,611	04:06
6	When The Clouds - The Place Where .mp3	12,542	05:21
7	Hammock - Mono No Aware.mp3	15,577	06:38
8	Frames - Don't Stay Here.mp3	20,2	08:37
9	Mogwai - Music for a forgotten future.mp3	53,452	22:48
	Total	120,092	55:18

Segue o resultado do primeiro testes, mostrados na tabela 2, utilizando um roteador para conectar o servidor ao cliente:

Tabela 2 Teste pela conexão com roteador externo

Nº	1º Reprodução		2º Reprodução		3º Reprodução		Média	
	Início (ms)	Buffer (ms)	Início (ms)	Buffer (ms)	Início (ms)	Buffer (ms)	Início (ms)	Buffer (ms)
1	1909	7835	1905	88661	1798	7789	1871	34762
2	1868	14864	1896	12878	1958	11922	1907	13221
3	2897	15825	2491	14360	2422	14355	2603	14847
4	3138	19061	1837	17813	2093	19064	2356	18646
5	2175	60109	2835	53817	2485	57424	2498	57117
6	4195	109252	2581	77567	2475	68509	3084	85109
7	2986	143994	3649	95649	2460	83387	3032	107677
8	2664	121720	2836	139854	2894	104912	2798	122162
9	3154	399302	2483	276533	2646	368882	2761	348239
	2276	891962	2501	777132	2359	736244	2546	801779,3

A tabela 3 mostra o resultado do segundo teste, utilizando o smartphone como roteador:

Tabela 3 Teste pela conexão com o roteador do smartphone

Nº	1º Reprodução		2º Reprodução		3º Reprodução		Média	
	Início (ms)	Buffer (ms)	Início (ms)	Buffer (ms)	Início (ms)	Buffer (ms)	Início (ms)	Buffer (ms)
1	435	1271	335	1293	407	1277	392	1280
2	421	1370	445	1262	600	1323	489	1318
3	425	1348	560	1362	466	1310	484	1340
4	291	1278	302	1310	312	1286	302	1291
5	404	2327	369	2327	528	2343	434	2332
6	382	3313	325	3315	613	3380	440	3336
7	351	4334	342	3310	444	4457	379	4034
8	364	4363	423	4393	414	5406	400	4721
9	492	33452	488	24475	495	18502	492	25476
Total	396	53056	399	43047	475	39284	423	45129

A tabela 4 mostra o resultado do teste pela conexão por USB:

Tabela 4 Testes utilizando a conexão via USB

Nº	1º Reprodução		2º Reprodução		3º Reprodução		Média	
	Início (ms)	Buffer (ms)	Início (ms)	Buffer (ms)	Início (ms)	Buffer (ms)	Início (ms)	Buffer (ms)
1	227	1265	273	1283	329	1254	276	1267
2	316	1311	255	1259	268	1260	280	1277
3	335	1328	256	1253	266	1264	286	1282
4	397	1389	248	1253	233	1224	293	1289
5	597	2546	258	2253	235	2232	363	2344
6	634	2597	302	2303	255	2247	397	2382
7	746	2699	318	3313	339	2319	468	2777
8	737	1358	288	3282	340	3334	455	2658
9	348	15356	442	14450	403	15405	398	15070
	482	29849	293	30649	296	30539	357	30346

Com base no teste efetuado foi possível observar que o meio menos direto de comunicação, apesar de ser o mais comum, é o que gera o menor desempenho e que o tamanho do arquivo não afeta criticamente o início da execução da música, apenas o tempo de download total do arquivo.

A execução com um roteador externo foi a que apresentou maior atraso entre as três comparações, com o tempo máximo de início da música em 4,195 segundos e o tempo médio de 2,546 segundos, em resumo, de acordo com os testes realizados, o tempo máximo de espera do usuário para o início de cada música no pior senário apresentado é de 4 segundos, sendo que após o início da música, não ouve travas no carregamento, com a música fluindo normalmente. O b*uffer completo* da música nesse cenário também foi bastante elevado, chegando a levar 348 segundos (5 minutos e 48 segundos) na maior música que possuía 22 minutos e 48 segundos, ainda assim, sem comprometer a reprodução.

A execução do cenário dois, com o smartphone como roteador, apresentou um desempenho bem superior ao primeiro cenário, já esperado, devido ao fato de ser uma conexão mais direta ao servidor, sem a necessidade de um componente intermediário entre eles. O tempo médio de início da música foi de menos 423 milissegundos e o tempo máximo foi de 613 milissegundos, em nenhum dos casos chegando a mais de 1 segundo, o que representa uma velocidade de quase sete vezes mais rápido que no primeiro cenário. Com relação ao tempo de *buffer* total, o tempo máximo de carregamento, para a mesma música do cenário um, foi de apenas 25 segundos, nesse caso, representando um desempenho treze vezes superior.

A execução do cenário três, utilizando a conexão USB, apresentou o melhor desempenho entre os três cenário, como esperado, devido a conexão direto via cabo, que possui um nível de ruídos e interferências bem menos aos modelos sem fio, porém foi bem inferior a diferença encontrada entre o cenário um e dois. O tempo médio de início de música foi de 357 milissegundos, representando um desempenho de 15% superior ao cenário dois, e o máximo de 737 milissegundos, que no caso acabou sendo superior ao tempo máximo do cenário dois.

Nos três cenários, apesar do tempo de início ter chegado a um nível elevado, superior a 4 segundos, após o início da reprodução da música, não houveram travas ou outros fatores que representassem problemas no ciclo de reprodução da música, chegando a um nível satisfatório no processo de reprodução.

4.4.4 – Custos do modelo proposto

Os custos relacionados ao desenvolvimento do produto estão relacionados principalmente ao tempo gasto de desenvolvimento do sistema, levando em consideração a duração efetiva do projeto de três meses, de agosto a meados de novembro, durante os finais de semana, dividindo o tempo entre a escrita da monografia e o desenvolvimento, estima-se que em cada semana foi gasto cerca de dez a doze horas de desenvolvimento. No total foram quatorze semanas, com o tempo aproximado de cento e quarenta horas de desenvolvimento.

Além do custo de desenvolvimento, também existiram os custos relacionados a ferramentas de desenvolvimento, como o notebook e o smartphone utilizados no processo, e uma licença do Visual Studio 2010, que apesar de ser uma licença fornecida gratuitamente pela faculdade, não permite o uso para desenvolvimento comercial do sistema.

Uma alternativa para reduzir os custos no desenvolvimento do sistema, com relação a licença utilizada do Visual Studio é a utilização do eclipse, ferramenta de desenvolvimento *opensource*, com um plugin de desenvolvimento para C#, como o *Emonic*.

Segue na tabela 5 os custos estimados de cada recurso utilizado no desenvolvimento do sistema:

Tabela 5 Custos de desenvolvimento

Recurso	Custo
Tempo de desenvolvimento	140 horas
Notebook WIN T546L+	R$ 1.700,00
Smatphone Samsung Galaxy SII	R$ 1.500,00
Licença Visual Studio 2010 Professional	R$ 2.414,67

4.4.5 – Requisitos Mínimos para a Utilização do Sistema

Para que o sistema possa ser utilizado, é necessário que o servidor seja um computador pessoal ou notebook, com o sistema operacional Windows 7 ou superior e para o funcionamento do cliente é necessário um smartphone com sistema operacional Android 4.0 ou superior. Para a conexão entre cliente e servidor é necessário que ambos estejam em uma mesma rede e que o endereço do servidor esteja devidamente configurado no cliente.

4.4.6 – Avaliação Global do Modelo

O projeto foi de grande valia para as práticas dos conhecimentos adquiridos nas disciplinas de lógica de programação, redes de computadores, processamento de sinais digitais, sistemas de comunicação, assim como conhecimentos profissionais.

O produto teve um bom desempenho nos testes apresentados, podendo ser utilizado de forma abrangente e podendo gerar até mesmo produtos comerciais de mesma funcionalidade.

CAPÍTULO 5 - CONCLUSÃO

5.1 - Conclusões

O projeto apresentado foi capaz de atingir os objetivos gerais de forma satisfatória. Como colocado na realização dos testes, não houve atraso o suficiente para atrasar a reprodução da música, onde apenas o atraso de 1 a 3 segundos para iniciar a reprodução da música pode ser considerado problemático.

Através do trabalho realizado foi possível adquirir bons conhecimentos nas linguagens de programação utilizadas, pouco conhecidas no início do projeto, porém existia uma boa base, o que facilitou na implementação dos códigos e na busca das soluções utilizadas.

Alguns pontos foram bastante críticos, como o desenvolvimento do modulo inicial de comunicação utilizando o WCF no servidor e a biblioteca no android e a pesquisa para chegar no HTTP como alternativa para o RTP, que levaria mais tempo de desenvolvimento e poderia impossibilitar a conclusão do trabalho a tempo.

5.2 - Sugestões para Trabalhos Futuros

Uma boa modificação possível neste projeto e permitir a reprodução de outros formatos de mídia, não apenas em questão de áudio, ampliando do MP3 para WAV, ou até mesmo FLAC, um dos formatos com melhor nível de qualidade de áudio, como também a reprodução de vídeo, em diversos formatos como, por exemplo, o MP4, ou o AVI.

Também seria possível com o projeto a implementação do protocolo RTP, tanto para o mesmo tipo de transmissão deste projeto, o streaming de mídia estática, como também a transmissão do streaming ao vivo. Juntamente com a implementação do RTP, poderia ser incluído uma análise comparativa de desempenho entre o modelo proposto nesse projeto, utilizando o HTTP, e o novo modelo implementado, definindo as vantagens e desvantagens de cada modelo e apresentando uma analise prática dos modelos.

REFERÊNCIAS

FARIA, J. O que é Streaming de Audio? **AudioTX Soluções de áudio e vídeo**, Novembro 2012. Disponivel em: <http://www.audiotx.com.br/?page_id=423>. Acesso em: 18 Novembro 2012.

GOOGLE. Activities. **Android Developers**, 25 Outubro 2012. Disponivel em: <http://developer.android.com/guide/components/activities.html>. Acesso em: 1 Novembro 2012.

GOOGLE. Application Fundamentals. **Android Developers**, 25 Outubro 2012. Disponivel em: <http://developer.android.com/guide/components/fundamentals.html>. Acesso em: 1 Novembro 2012.

GOURLEY, D.; TOTTY, B. **HTTP:** The Definitive Guide. [S.l.]: O'Reilly & Associates, Inc., 2002.

J.I, H. Do Anything With ID3. **Code Project**, 22 mar. 2007. Disponivel em: <http://www.codeproject.com/Articles/17890/Do-Anything-With-ID3>. Acesso em: 17 set. 2012.

KUROSE , J. F.; ROSS, K. W. **Computer Networking:** A Top-Down Approach (5th Edition). [S.l.]: Addison-Wesley, 2009.

LEE, W.-M. **Introdução ao Desenvolvimento de Aplicativos para o Android**. Rio de Janeiro: Ciência Moderna, 2011.

MICROSOFT. MSDN. **MSDN**, 22 Setembro 2012. Disponivel em: <http://msdn.microsoft.com/en-us/library/hh425099.aspx>. Acesso em: 22 Setembro 2012.

MUSMANN, H. G. **The ISO Audio Coding Standard**. Global Telecommunications Conference, 1990, and Exhibition. 'Communications: Connecting the Future', GLOBECOM '90., IEEE. West Germany: IEEE. 1990. p. 7.

MUSMANN, H. G. Genesis of the MP3 audio coding standard. **IEEE Transactions on Consumer Eletronics**, p. 7, 2003.

MUSMANN, H. G. Genesis of the MP3 audio coding standard. **IEEE Transactions on Consumer Eletronics**, p. 7, 2006.

PAHLAVAN, ; LEVESQUE, A. H. **Wireless Information Networks**. Canada: John Wiley & Sons, Inc., 2005.

PERKINS, C. **RTP:** Audio and Video for the Internet. [S.l.]: Addison Wesley, 2003.

SCHILDT, H. **C# 4.0:** The Complete Reference. [S.l.]: Mc Graw Hill, 2010.

TOTTY, B. et al. **HTTP:** The Definitive Guide. [S.l.]: O'Reilly Media, 2002.

APÊNDICE

APÊNDICE A – Código Fonte do Servidor

Constam nesse apêndice o código fonte da solução implementada na parte do servidor, sendo cada parte um arquivo separado.

A1 – Server.cs (Classe principal do programa)

```
/* Classe principal do sistema, responsável pela inicialização do sistema, configuração inicial dos componentes, assim como configuração dos eventos dos elementos disponíveis ao usuário
*/
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO;
using System.ServiceModel;
using ServerStreaming.http;
using System.Threading;
using System.Net;

namespace ServerStreaming
{
   public partial class Server : Form
   {
      // Constante da porta utilizada na conexão HTTP
      const int HTTP_PORT = 8080;
      // Objeto principal da aplicação
      PlaylistController controler;
      // Objeto do service WCF
      ServiceHost host;
```

```
// Objeto do service HTTP
HttpServer serviceHTTP;
/** Função responsável por buscar o IP do servidor **/
public string GetIPAddress()
{
   string strHostName = System.Net.Dns.GetHostName();
   IPHostEntry ipHostInfo = Dns.Resolve(Dns.GetHostName());
   IPAddress ipAddress = ipHostInfo.AddressList[0];

   return ipAddress.ToString();
}

/** Função que Inicializa os serviços do sistema **/
public void InicialiarSistema()
{

   // Pega a instância do controller
   controler = PlaylistController.getInstance();
   // Adiciona a lista de playlists no listbox da playlist
   foreach (var playlist in controler.playlists)
   {
      PlaylistBox.Items.Add(playlist.name);
   }
   // Inicializa serviço WEB Service WCF
   host = new ServiceHost(typeof(communicate.communicate), new Uri[] { });
   host.Open();

   // Inicializa HTTP Server para streaming de audio
   serviceHTTP = new HttpServer(HTTP_PORT);
   Thread thread = new Thread(new ThreadStart(serviceHTTP.listen));
   thread.Start();

   bxEndereco.Text = GetIPAddress();
}
```

```
        /** Função de inicialização do programa **/
        public Server()
        {
            // Executa a criação dos componentes gráficos da janela
            InitializeComponent();

            // Executa a inicialização dos componentes lógicos do sistema
            InicialiarSistema();

        }
        #region controle;

        /** Função de adição de nova playlist **/
        public int AdicionarPlaylist(String nome)
        {
            // Adiciona a playlist a lista de controle de playlists
            int ret = controler.AdicionarPlaylist(nome);
            // Verifica se o retorno do controle foi 1 (OK) se sim, adiciona a playlist na lista de
playlist da janela
            if (ret == 1)
            {
                PlaylistBox.Items.Add(nome);
            }
            //Retorna o resultado da inserção
            return ret;
        }

        /** Função que abre a janela de criação de nova playlist **/
        private void OpenNewPlayList()
        {
            // Cria a janela e passa o objeto da janela atual (a principal) para a janela criada
            // Isso é feito para que a janela aberta possa usar as funções da janela principal
            (new NovaPlaylist(this)).ShowDialog();
        }
```

```
        /** Função que remover uma playlist especifica da lista **/
        private void RemoverPlaylist(int index)
        {
            // Verifica se o número da lista é válido, se for -1, significa que nenhuma playlist foi
selecionada
            if (PlaylistBox.SelectedIndex != -1)
            {
                //Executa a função de remover a playlist e verifica se o retorno foi igual a 1 (OK)
                if (controler.Remover(index) == 1)
                {
                    //Se a playlist foi removida do controler, ela é removida a lista da janela
                    PlaylistBox.Items.RemoveAt(index);
                }
            }
        }
        /** Função para abrir a caixa de dialogo **/
        public void OpenFileDialog()
        {
            try
            {
                // Verifica se tem alguma playlist selecionada
                if (controler.index_atual == -1)
                {
                    //Caso não tenha nenhuma playlist selecionada joga uma exceção informando
uma mensagem
                    throw new Exception("É necessário selecionar uma playlist.");
                }
                // Abre a caixa de dialogo
                DialogResult result = OpenMusicFiles.ShowDialog();
                // Após a caia de dialogo ser fechada, verifica se o resultado foi OK, caso contrario
interrompe a função
                if (result != DialogResult.OK) return;
                // Loop para incluir as músicas selecionadas no sistema
```

```
            foreach (var arquivo in OpenMusicFiles.FileNames)
            {
               // Cria e Inicializa o objeto do tipo Musicas, passando o path do arquivo
               Musicas musica = new Musicas(arquivo);
               //Adiciona a musica ao controle, que colocará a musica na playlist selecionada
               if (controler.AdicionarMusica(musica) == 1)
               {
                  //Executa a função para inserir a musica no listbox, se ela tiver sido adicionada
no controler
                  ShowMusic(musica);
               }
            }
         }
         // Exception de falha
         catch (Exception ex)
         {
            // Exibe um alerta na tela com a mensagem do erro informado
            MessageBox.Show(ex.Message);
         }
      }

      /** Função que exibe as músicas da playlist selecionada **/
      private void ShowMusicList()
      {
         // Limpa o listbox de músicas
         MusicBox.Items.Clear();
         // verifica se o index_atual do controler é diferente de nulo, ou seja, se existe alguma
playlist selecionada
         if (controler.index_atual != null)
         {
            // Loop para incluir as músicas da playilist, retornadas pela função getMusicas do
controler
            foreach (var musica in controler.getMusicas())
            {
```

```
                // Executa a função que coloca a musica no listbox, passando o objeto musica
como parametro
                ShowMusic(musica);
            }
        }
    }
    /** Função que adiciona a música ao listbox de músicas **/
    private void ShowMusic(Musicas musica)
    {
        // Insere a musica na lista, colocando o artista e o titulo
        MusicBox.Items.Add(musica.artista + " - " + musica.titulo);
    }
    /** Função que remove a música da playlist **/
    private void RemoverMusica(int index)
    {
        // Verifica se o número da música é valido
        if (index != -1)
        {
            // Remove a música do controler
            controler.RemoverMusica(index);
            // Remove a música da listbox de músicas
            MusicBox.Items.RemoveAt(index);
        }
    }
    #endregion;


    /** Evento onclick do botão "Adicionar" da playlist **/
    private void bt_adicionar_playlist_Click(object sender, EventArgs e)
    {
        OpenNewPlayList();
    }
    /* Função de evento onchange da listbox da playlist **/
    private void PlaylistBox_SelectedIndexChanged(object sender, EventArgs e)
```

```
        {
            // Altera a lista selecionada no controler
            controler.select(PlaylistBox.SelectedIndex);
            // Atualiza a lista de músicas
            ShowMusicList();
        }

        /** Função de evento do botão de adicionar músicas **/
        private void bt_adicionar_musica_Click(object sender, EventArgs e)
        {
            // Função que abre a caixa de dialogo
            OpenFileDialog();
        }
        /** Função de evento do botão de remover da lista de músicas **/
        private void bt_remover_musica_Click(object sender, EventArgs e)
        {
            //Executa a função de remover passando o número da música selecionada
            RemoverMusica(MusicBox.SelectedIndex);
        }

        /** Evento onclic do botão "Remover" playlist **/
        private void rm_playlist_Click(object sender, EventArgs e)
        {
            //Executa a função de remover playlist, enviando o número da playlist selecionada
            RemoverPlaylist(PlaylistBox.SelectedIndex);
        }


    }
}
```

A.2 – NovaPlaylist.cs

```
/** Classe da janela de criação de nova playlist, utilizada para exibir uma caixa de texto para a inserção da playlists **/
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ServerStreaming
{
   public partial class NovaPlaylist : Form
   {
      // Objeto para referenciar a janela Server
      Server parent;
      /* Função de Inicialzação da janela */
      public NovaPlaylist(Server sender)
      {
         InitializeComponent(); // Função que inicializa o componente
         parent = sender; // Pega o sender, que é a janela pai que chamou o componente NovaPlaylist, e salva sua referencia em parent
      }

      /* Função para adicionar a nova playlist, ela valida as informações do campo de texto da janela e,
       *  em caso de diferente de vazio, envia para a janela principal o nome da nova playlist e espera a resposta de confirmação.
       *  Caso a resposta seja 1, significa que a playlist foi adicionada, caso -1, existe outra playlist com o mesmo nome
       *   e diferente, houve uma falha desconhecida na adição da playlist.
       */
      public void AddPlaylist()
```

```
        {
            try
            {
                // Valida se o nome é diferente de vazio
                if (play_name.Text == "")
                    throw new Exception("Informe o nome da nova playlist");

                // Envia o nome da playlist para a janela principal e compara o resultado retornado;
                switch (parent.AdicionarPlaylist(play_name.Text))
                {
                    // Playlist inserida
                    case 1: break;
                    // Nome informado já existe
                    case -1: throw new Exception("Já existe outra playlist com esse nome.");
                    // Falha desconhecida
                    default: throw new Exception("Falha ao criar nova playlist.");
                }
                // Fecha a janela, caso nenhum exception de erro seja disparado.
                Close();
            }
            catch (Exception exp)
            {
                //Exibe o exception de erro.
                MessageBox.Show(exp.Message);
            }
        }
        // Evento de onclick do botão adicionar
        private void bt_adicionar_Click(object sender, EventArgs e)
        {
            AddPlaylist();
        }
        // Evento de keypress do campo de texto, quando a tecla enter é pressionada ele executa a
função de adicionar a playlist
        private void play_name_KeyPress(object sender, KeyPressEventArgs e)
```

```
        {
            if (e.KeyChar == (char)13)
            {
                e.Handled = true;
                AddPlaylist();
            }
        }
        // Evento de quando a janela é exibida, para colocar o cursor no campo de texto.
        private void NovaMusica_Shown(object sender, EventArgs e)
        {
            play_name.Focus();
        }
    }
}
```

A.3 - PlaylistController.cs

```
/** Função principal de controle, nela ficam concentradas os dados de músicas e playlists, assim como a gestão desses itens, tal como adicionar/remover músicas e playlists, e salvar as informações do sistema **/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Xml;
using System.IO;

namespace ServerStreaming
{
    /**  Classe principal de controle de playlist **/
    class PlaylistController
    {
        // Objeto estático controller, que é instanciado apenas uma vez e referenciado em todos os objetos de mesma classe
        public static PlaylistController controller;
```

```
// Lista de playlists
public List<Playlist> playlists;
// Playlist atual (playlist selecionada para gerenciamento)
Playlist playlist_atual;
// Index da playlist atual na lista de playlists
public int index_atual = -1;

/** Função que retorna a instância estática da classe **/
public static PlaylistController getInstance()
{
   // Verifica se a instância já foi inicializada, inicializando o objeto se não
   if (controller == null)
   {
      controller = new PlaylistController();
   }
   // Retorna a instância estática
   return controller;
}
/** Funcão de inicialização, ela é privada de modo a não permitir incialização de objeto externo a classe, para que só possa ser instanciada uma unica vez **/
private PlaylistController()
{
   // Inicializa a lista de playlists
   playlists = new List<Playlist>();
   // Carrega a lista de playlists salva
   CarregarLista();
}
/** Funcão que adiciona uma nova playlist no controler **/
public int AdicionarPlaylist(String nome, long ID = 0)
{
   // Consulta na lista de playlists que verifica se não existe uma playlist de mesmo nome
   var Query = from playlist in playlists where playlist.name == nome select playlist;
   // Verifica se foi encontrado alguma playlist e, se sim, retorna -1
```

```
            if (Query.Count() == 1)
                return -1;
            //Inicializa nova playlist e adiciona na lista do controler
            Playlist nova_playlist = new Playlist(nome, ID);
            playlists.Add(nova_playlist);
            // Salva o XML da lista de playlist
            Save();
            return 1;
        }
        /** Função para selecionar uma determinada playlist para gerenciamento**/
        public bool select(int index)
        {
            // Verifica se o index é válido para a seleção
            if (index != -1 && index < playlists.Count())
            {
                // Seta a playlist atual no controller e retorna true (sucesso)
                playlist_atual = playlists[index];
                index_atual = index;
                return true;
            }
            else
            {
                // Caso seja um index inválido, limpa as referencias da playlist atual e retorna false
(falha)
                index_atual = -1;
                playlist_atual = null;
                return false;
            }
        }
        /** Função para remover uma playlist da lista **/
        public int Remover(int index)
        {
            // Apaga XML da playlist
            playlists[index].ApagarXML();
```

```
    // Remove a playlist
    playlists.RemoveAt(index);
    // Salva a playlist
    Save();
    // Retorna 1 para indicar sucesso
    return 1;
}
/** Função para adicionar uma nova música na playlist atual */
public int AdicionarMusica(Musicas musica)
{
    // Adiciona na playlist e retorna o resultado
    return playlist_atual.AdicionarMusica(musica);
}
/** função para remover uma música da playlist **/
public void RemoverMusica(int index)
{
    // Remove a música da playlist atual
    playlist_atual.RemoverMusica(index);
}
/** Função que retorn a lista de músicas de uma playlist de acordo com o ID **/
public List<Musicas> getMusicas(String playlistID)
{
    // Busca a playlist com o ID especifico
    Playlist lista = getPlaylistById(playlistID);
    if (lista != null)
    {
        // Retorna a lista de músicas, caso tenha sido encontrada
        return lista.musicas;
    }
    // Retorna null caso a playlist não seja encontrada
    return null;
}
/** Função que retorna a lista de músicas da playlist atual **/
```

```
        public List<Musicas> getMusicas()
        {
            // Verifica se tem alguma playlist selecionada e retorna a lista de músicas da playlist,
ou uma lista vazia caso não tenha nenhuma selecionada
            if (playlist_atual != null)
                return playlist_atual.musicas;
            else return new List<Musicas>();
        }
        /** Função que gera o XML com a lista de playlists **/
        public String XML()
        {
            // Inicializa o objeto xml
            XmlDocument xml = new XmlDocument();
            // Cria a tag Playlist no xml
            XmlElement root = xml.CreateElement("Playlists");
            // Insere a tag criada no objeto principal do xml
            xml.AppendChild(root);
            // Loop para adicionar as playlists na tag Playlists
            foreach (var playlist in playlists)
            {
                XmlElement xml_node= xml.CreateElement("playlist");
                xml_node.SetAttribute("ID", playlist.ID);
                xml_node.AppendChild(xml.CreateTextNode(playlist.name));
                root.AppendChild(xml_node);

            }
            // Retorna o xml gerado
            return xml.OuterXml;
        }
        /** Função que salva o XML do controler **/
        public void Save()
        {
            // Gera a string com o path onde será salvo a lista de playlists
            string mydocpath = Path.Combine(Environment.CurrentDirectory, "xml");
```

```
            // Verifica se o path não existe, criando se necessário
            if (!Directory.Exists(mydocpath))
            {
                Directory.CreateDirectory(mydocpath);
            }
            // Salva o XML em disco
            using (StreamWriter xml_save = new StreamWriter(mydocpath + "\\playlists.xml"))
            {
                xml_save.Write(XML());
            }
        }
        /** Função que carrega a lista de playlists do XML **/
        public void CarregarLista()
        {
            // Gera o path do XML da lista
            string xmlpath = Path.Combine(Environment.CurrentDirectory, "xml",
"playlists.xml");
            // Verifica se o xml existe e se sim, efetua a leitura do xml
            if(File.Exists(xmlpath))
            {
                XmlDocument xmlDoc = new XmlDocument();
                xmlDoc.Load(xmlpath);
                // Lê o XML e salva as playlists salvas na lista do controler
                XmlNodeList xmlList = xmlDoc.GetElementsByTagName("playlist");
                for (int i = 0; i < xmlList.Count; i++)
                {
                    string name = xmlList[i].InnerText;
                    long ID = Convert.ToInt64( xmlList[i].Attributes["ID"].InnerText);
                    AdicionarPlaylist(name, ID);
                }
            }
        }
        /** Função que busca uma playist pelo ID **/
        public Playlist getPlaylistById(string playlistID)
```

```
        {
            // Consulta na lista de playlists
            var Query = from playlist in playlists where playlist.ID == playlistID select playlist;
            if (Query.Count() == 1)
            {
                // Pega a lista de playilst encontrada e retorna a playlist
                List<Playlist> teste = Query.ToList();
                return teste[0];
            }
            // Retorna NULL caso nenhuma playlista seja encontrada
            return null;
        }
        /** Função que retorna o path de uma música de uma playlsit pelo ID da música e da
playlsit **/
        public String getMusicPath(string playlistID, string MusicaID)
        {
            // Busca a playlist
            Playlist lista = getPlaylistById(playlistID);
            if (lista != null)
            {
                // Busca a música da playlist encontrada
                Musicas musica = lista.getMusicaById(MusicaID);
                if (musica != null)
                {
                    // retorna o path da música encontrada
                    return musica.path_file;
                }
            }
            // Retorna null caso nenhuma música seja encontrada
            return null;
        }
    }
}
```

A.4 - Playlist.cs

```
/** Classe responsável por guarder e gerenciar uma playlist **/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data;
using System.Xml;
using System.IO;

namespace ServerStreaming
{
   /** Função responsavel por guardar e gerenciar a playlist **/
   class Playlist
   {
      // Lista de músicas da playlist
      public List<Musicas> musicas;
      // Nome da playlist
      public String name;
      // ID da playlist
      public String ID;

      private static long last_id = 0;

      /** Funcão de inicialização da playlist **/
      public Playlist(String par_name, long par_ID = 0)
      {
         // Gera o ID da playlist com base no nome e avança na lista geral de ids
         this.ID = par_ID != 0 ? par_ID.ToString() : last_id.ToString();
         last_id = (par_ID > last_id ? par_ID : last_id) + 1;


         // Inicializa a lista de músicas
         musicas = new List<Musicas>();
```

```
        // Salva o nome da playlist
        name = par_name;
        // Carrega as músicas se o XML da playlist existir
        CarregarMusicas();
    }
    /** Função responsavel por adicionar a música na playlist **/
    public int AdicionarMusica(Musicas musica){
        // Verfica se a música já existe na playlist, com base no ID
        if (getMusicaById(musica.ID) == null)
        {
            // Adiciona a música na playlist
            musicas.Add(musica);
            // Salva/Atualiza o XML da playlist
            Save();
            // Retorna sucesso.
            return 1;
        }
        else
        {
            // Retorna -1 que indica que a música já existe na playlist
            return -1;
        }
    }
    /** Função que retorna a quantidade de músicas na playist **/
    public int QtdMusic()
    {
        return musicas.Count;
    }
    /** Função que remove uma música da playlist, através da posição dela na fila **/
    public void RemoverMusica(int index)
    {
        // Verifica se o index da lista é valido, se está dentro da quantidade de músicas da
playlist
        if (index < musicas.Count)
```

```
    {
        // Remove a música da playlist e atualia o XML
        musicas.RemoveAt(index);
        Save();
    }
}
/** Função que gera o XML da playlist **/
public String XML()
{
    // Inicializa o objeto xml
    XmlDocument xml = new XmlDocument();
    // Cria a tag principal do xml
    XmlElement root = xml.CreateElement("Musicas");
    // Adiciona a tag principal no xml
    xml.AppendChild(root);
    // Loop para inserir a lista de músicas na tag Musicas no xml
    foreach (var musica in musicas)
    {
        XmlElement xml_node = xml.CreateElement("musica");
        xml_node.SetAttribute("ID", musica.ID);
        xml_node.AppendChild(xml.CreateTextNode(musica.path_file));
        root.AppendChild(xml_node);
    }
    // Retorna o xml gerado
    return xml.OuterXml;
}

/** Função que salva o xml da playlist **/
public void Save()
{
    // Gera o path da onde será salvo a playlist
    string mydocpath = Path.Combine(Environment.CurrentDirectory, "xml", "playlists");
    // Verifica se o diretorio não existe e cria se necessário.
    if (!Directory.Exists(mydocpath))
```

```
            {
                Directory.CreateDirectory(mydocpath);
            }
            // Salva o XML gerado no path em disco
            using (StreamWriter xml_save = new StreamWriter(mydocpath + "\\" + ID + ".xml"))
            {
                xml_save.Write(XML());
            }
        }
        /** Função que carrega as músicas do XML da playlist **/
        public void CarregarMusicas()
        {
            // Gera a string contendo o path do arquivo XML
            string xmlpath = Path.Combine(Environment.CurrentDirectory, "xml", "playlists", ID
+ ".xml");
            // Verifica se o arquivo existe, carregando os dados se existir
            if (File.Exists(xmlpath))
            {
                // Inicializa o xml
                XmlDocument xmlDoc = new XmlDocument();
                // Carrega o aquiro no objeto
                xmlDoc.Load(xmlpath);

                // Pega todas as tags musica do XML
                XmlNodeList xmlList = xmlDoc.GetElementsByTagName("musica");
                // Loop para adicionar as músicas na playlist
                for(int i = 0; i < xmlList.Count; i++)
                {
                    // pega o path e ID da música do xml
                    string path = xmlList[i].InnerText;
                    long musica_id = Convert.ToInt64( xmlList[i].Attributes["ID"].InnerText);
                    // Inicializa o objeto musica passando o path como parametro
                    Musicas musica = new Musicas(path, musica_id);
                    // Adiciona a música na playlist
```

```
                AdicionarMusica(musica);
            }
        }
    }
    // Apaga o xml com as músicas da playlist
    public void ApagarXML()
    {
        // Gera a string contendo o path do arquivo XML
        string xmlpath = Path.Combine(Environment.CurrentDirectory, "xml", "playlists", ID
+ ".xml");
        File.Delete(xmlpath);
    }
    /** Função que busca uma música pelo ID **/
    public Musicas getMusicaById(String MusicaID)
    {
        // Consulta na lista de músicas para buscar a música com o ID especificado
        var Query = from musica in musicas where musica.ID == MusicaID select musica;
        // Verifica se a consulta retornou uma música
        if (Query.Count() == 1)
        {
            // Retorna a música encontrada
            List<Musicas> musica = Query.ToList();
            return musica[0];
        }
        // Retorna null, caso nenhuma música tenha sido encontrada
        return null;
    }
  }
}
```

A.5 - Musicas.cs

```
/**  Classe responsável por obter e guardar as informações da música da playlist **/
using System;
using System.Collections.Generic;
```

```
using System.Linq;
using System.Text;

namespace ServerStreaming
{
   /** Classe responsavel por armazenas as informações da música **/
   class Musicas
   {
      // ID da musica
      public String ID;
      // Nome do arquivo de música
      public String file_name;
      // Caminho onde está salvo a música
      public String path_file;
      // Titulo da música
      public String titulo;
      public String album;
      // Artista/Banda da música
      public String artista;
      // Long de controle de ID's
      private static long last_id = 0;

      /** Função de inicialização do objeto, responsável por buscar as informações da música
**/
      public Musicas(String _pathFile, long par_ID = 0)
      {
         // Salva o caminho da imagem no objeto
         path_file = _pathFile;
         //-- Leitura das informações da música
         ID3.ID3Info tag = new ID3.ID3Info(path_file, true);
         // -- Busca o titulo e artitas da música com base na leitura da tag ID3 do arquivo mp3
         titulo = (!string.IsNullOrEmpty(tag.ID3v1Info.Title) ? tag.ID3v1Info.Title :
tag.ID3v2Info.GetTextFrame("TIT2")).Trim();
```

```
            artista = (!string.IsNullOrEmpty(tag.ID3v1Info.Artist) ? tag.ID3v1Info.Artist :
tag.ID3v2Info.GetTextFrames("TOPE", "TPE1", "TPE2")).Trim();
            album = (!string.IsNullOrEmpty(tag.ID3v1Info.Album) ? tag.ID3v1Info.Album :
tag.ID3v2Info.GetTextFrame("TALB")).Trim();
            // Pega o nome do arquivo de música
            file_name = tag.FileName;

            // Gera o ID da playlist com base no nome e avança na lista geral de ids
            this.ID = par_ID != 0 ? par_ID.ToString() : last_id.ToString();
            last_id = (par_ID > last_id ? par_ID : last_id) + 1;
        }
    }
}
```

A.6 – Icommunicate.cs

```
/** Interface do sistema, nela são definidos os métodos disponíveis no webservice, assim
como as interfaces de dados que são enviadas. **/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace ServerStreaming.communicate
{
    [DataContract]
    public class cplaylist
    {
        [DataMember]
        public String ID;
        [DataMember]
        public String Name;
    }
    [DataContract]
```

```
// Resposta da consulta de playists
public class playlistResponse
{
   [DataMember]
   // Codigo de retorno, caso diferente de 0, significa erro na consulta
   public int code;
   [DataMember]
   // Lista de playlists
   public List<cplaylist> playlist;
   // Inicializador da classe
   public playlistResponse()
   {
      // Inicializa a lista de plalists
      playlist = new List<cplaylist>();
   }
}
[DataContract]
public class cmusicas
{
   [DataMember]
   public String ID;
   [DataMember]
   public String titulo;
   [DataMember]
   public String artista;
   [DataMember]
   public String descricao;
}
[DataContract]
// Classe de resposta para o servidor
public class musicaResponse
{
   [DataMember]
   // Codigo de erro
```

```
        public int code;
        [DataMember]
        // Lista de musicas
        public List<cmusicas> musicas;
        // Inicializador da classe
        public musicaResponse()
        {
            // Inicializa a lista de musicas
            musicas = new List<cmusicas>();
        }
    }
    [ServiceContract]
    public interface Icommunicate
    {
        [OperationContract]
        /** Função que retorna a lista de playlists configuradas no servidor **/
        playlistResponse getPlaylist();
        [OperationContract]
        /** Função que retorna a lista de músicas de uma playlist configuradas no servidor **/
        musicaResponse getMusica(String PlaylistID);
    }
}
```

A.7 – communicate.cs

```
/** Classe responsável pela implementação dos metodos da interface Icommunicate para a comunicação com o cliente**/
using System;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.Text;

namespace ServerStreaming.communicate
{
```

```
public class communicate : Icommunicate
{
   public playlistResponse getPlaylist()
   {
      // Pega a instância do controller
      PlaylistController controller = PlaylistController.getInstance();
      // Cria e instância o objeto de resposta
      playlistResponse response = new playlistResponse();
      try
      {
         // preenche a lista de playlists para enviar ao cliente
         foreach (var playlist in controller.playlists)
         {
            cplaylist play = new cplaylist();
            play.ID = playlist.ID;
            play.Name = playlist.name;
            response.playlist.Add(play);
         }
         // Seta o código de sucesso
         response.code = 0;
      }
      catch (Exception e)
      {
         // Seta o código de erro
         response.code = -1;
      }
      // Envia a resposta
      return response;
   }
   public musicaResponse getMusica(String PlaylistID)
   {
      // Pega a instância do controller
      PlaylistController controller = PlaylistController.getInstance();
```

```
// Cria e instância o objeto de resposta
musicaResponse response = new musicaResponse();
try
{
   // Verifica se foi enviado um PlaylistID
   if (PlaylistID == null)
   {
      // Seta o código de erro
      response.code = -1;
   }
   else
   {
      // Busca a lista de músicas
      List<Musicas> musicas = controller.getMusicas(PlaylistID);

      // Verifica se foram encontradas as músicas
      if (musicas == null)
      {
         // Seta o código de erro
         response.code = -1;
      }
      else
      {
         // Preenche a lista de músicas da playlist
         foreach (Musicas musica in musicas)
         {
            cmusicas temp = new cmusicas();
            temp.artista = musica.artista;
            temp.titulo = musica.titulo;
            temp.descricao = musica.artista + " - " + musica.titulo;
            temp.ID = musica.ID;
            response.musicas.Add(temp);
         }
      }
```

```
            }
        }
        catch (Exception error)
        {
            // Seta o código de erro em caso de alguma exceção disparada
            response.code = -1;
        }
        // Envia a resposta
        return response;
    }
  }
}
```

A.8 – app.config

```
<!—Arquivo XML de configuração para o service de webservice da aplicação -->
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="ServerStreaming.communicate.communicate"
behaviorConfiguration="wsdlConfiguration">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:9000"/>
          </baseAddresses>
        </host>
        <endpoint
          binding="basicHttpBinding"
          contract="ServerStreaming.communicate.Icommunicate"  />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="wsdlConfiguration">
```

```
        <serviceMetadata httpGetEnabled="true" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
 </system.serviceModel>
</configuration>
```

A.9 – HttpServer.cs

```
/** Classe principal de implementação do protocolo HTTP, responsável pela transmissão da
música para o cliente **/
using System;
using System.Collections;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Threading;

namespace ServerStreaming.http
{
   /** Class Abstrata de criação de Servidor HTTP **/
   public class HttpServer
   {
      // Porta de conexão
      protected int port;
      // Listener TCP
      TcpListener listener;
      // Flag indicando se a conexão está ativa
      bool is_active = true;

      /** Construtor da classe **/
      public HttpServer(int port)
      {
         // Salva a porta confirada
         this.port = port;
      }
      /** Metotodo listener, tem a função de aguardar uma requisição http **/
      public void listen()
      {
         // Inicia o listener
         listener = new TcpListener(port);
         listener.Start();
         while (is_active)
         {
            // Pega a requisição do cliente
            TcpClient s = listener.AcceptTcpClient();
```

```
            // Cria o objeto que irá tratar a requisição
            HttpProcessor processor = new HttpProcessor(s, this);
            //Cria uma thread para tratar a requisição do processor
            Thread thread = new Thread(new ThreadStart(processor.process));
            // Inicia a thread
            thread.Start();
            // Aguarda um intervalo antes de continuar o loop
            Thread.Sleep(1);
        }
    }

    public void handleRequest(HttpProcessor p)
    {
        /** Pega a instância do controller **/
        PlaylistController controller = PlaylistController.getInstance();
        /** pega a url requisitada pelo cliente **/
        String request_url = p.http_url;

        /** Quebra a string em partes para buscar qual é a playlist e qual a música solicitada
**/
        string[] RequestPart = request_url.Split('/');
        try
        {
            /** Verifica se a url está no padrão solicitado **/
            if (RequestPart.Length != 3)
            {
                throw new Exception();
            }
            // Busca a música solicitada
            String path = controller.getMusicPath(RequestPart[1], RequestPart[2]);
            // verifica se foi encontrada
            if (path == null)
            {
                throw new Exception();
            }
            //verifica se a música existe no servidor
            if (File.Exists(@path))
            {
                // Lê os bytes da musica
                byte[] fileBytes = File.ReadAllBytes(@path);
                // Busca as informações do arquivo, para buscar o tamanho do arquivo
                FileInfo info = new FileInfo(@path);
                // Seta o reader de sucesso, enviando o enconde type da música
                p.writeSuccess("audio/mpeg", info.Length);
                // transmite a música para o cliente
                p.outputStream.Write(fileBytes, 0, fileBytes.Length);
            }
            else
            {
                throw new Exception();
            }
```

```
            }
            catch (Exception e)
            {
               // envia a mensagem de falha caso qualquer erro ocorra no processo.
               p.writeFailure();
            }

         }
      }
   }
```

A.10 – HttpProcessor.cs

```
/** Classe reponsável por processar uma requisição HTTP **/

using System;
using System.Collections;
using System.IO;
using System.Net;
using System.Net.Sockets;
using System.Threading;

namespace ServerStreaming.http
{
   public class HttpProcessor
   {
      /** objetos de conexão */
      public TcpClient socket;
      public HttpServer srv;
      public NetworkStream Net;

      /** objetos de entrada e saída da requisiçao **/
      public Stream inputStream;
      public Stream outputStream;

      /** String contendo a url solicitada pelo usuário**/
      public String http_url;

      /** Constructor da classe HttpProcessor, que recebe o objeto de conexão com o socket e
o objeto do servidor HTTP**/
      public HttpProcessor(TcpClient s, HttpServer srv)
      {
         this.socket = s;
         this.srv = srv;
      }
      /** Função responsável por ler uma linha do streaming de requisição do cliente*/
      private string streamReadLine(Stream inputStream)
      {
         int next_char;
         string data = "";
```

```
            while (true)
            {
               next_char = inputStream.ReadByte();
               if (next_char == '\n') { break; }
               if (next_char == '\r') { continue; }
               if (next_char == -1) { Thread.Sleep(1); continue; };
               data += Convert.ToChar(next_char);
            }
            return data;
         }
         /** Função responsável por processar a requisição**/
         public void process()
         {
            try
            {
               /** Inicializa os objetos de entrada e saída **/
               inputStream = new BufferedStream(socket.GetStream());
               outputStream = new BufferedStream(socket.GetStream());
               try
               {
                  // Executa a função que irá separá os elementos da requisição, pegando assim a
url requisitada **/
                  parseRequest();
                  // envia para o objeto de serviço o objeto processado da requisição
                  srv.handleRequest(this);
               }
               catch (Exception e)
               {
                  // Envia uma mensagem de falha para o cliente caso algum erro aconteça
                  writeFailure();
               }
               // libera os objetos da conexão e fecha o socket
               outputStream.Flush();
               inputStream = null; outputStream = null;
               socket.Close();
            }
            catch (Exception e)
            {

            }
         }
         // Processa o cabeçalho da requisição
         public void parseRequest()
         {
            String request = streamReadLine(inputStream);
            string[] tokens = request.Split(' ');
            if (tokens.Length != 3)
            {
               throw new Exception("invalid http request line");
            }
            http_url = tokens[1];
```

```
        }
        // Escreve o cabeçalho de resposta em caso de sucesso
        public void writeSuccess(String ContentType = "text/html", long length = 0)
        {
            String sheader = "HTTP/1.0 200 OK\n"
                        + "Content-Type: " + ContentType + "\n"
                        + "Content-Transfer-Encoding: binary\n"
                        + "Content-Length:" + length.ToString() + "\n"
                        + "Connection: close\n\n";

            byte[] header = Util.StrToByteArray(sheader);
            outputStream.Write(header, 0, header.Length);
        }
        // Escreve o cabeçalho de falha em caso de erro
        public void writeFailure()
        {
            byte[] header = Util.StrToByteArray("HTTP/1.0 404 File not found\nConnection:
close\n\n");
            outputStream.Write(header, 0, header.Length);
        }
    }
}
```

APÊNDICE B – Código Fonte do Cliente Android

Constanam nesse apêndice os códigos utilizados no desenvolvimento do aplicativo cliente desenvolvido para dispositivos móveis com sistema operacional android.

B.1 – MainActivity.java

```
/** Classe responsável por gerenciar a tela inicial do sistema, com a lista de playlists**/
package com.emerson.streaming.client;


import java.util.ArrayList;


import android.app.Activity;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.view.Menu;
import android.view.MenuInflater;
```

```
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.ProgressBar;
import android.widget.TextView;

import com.emerson.streaming.controller.controller;
import com.emerson.streaming.controller.playlist;

public class MainActivity extends Activity {
        // instância do controler principal
   controller control = controller.getInstance();

   // Handle para atualizar view após execução da thread
   public Handler handler;
   // Flag que indica se o sistema já foi inicializado
   static boolean inicializado= false;

   // Função base da activity principal
   @Override
   public void onCreate(Bundle savedInstanceState) {
        // Inicializam componentes graficos
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_main);

      control.context = getBaseContext();
      control.getConfiguracao();

      // Caso inicializado = false, executa rotina para buscar as playlists
      if(!inicializado)
         buscarPlaylist();
```

```
        // Caso inicializado = true, executa a rotina que exibe as playlists
    else
        showPlaylist();
    inicializado = true;
}
/** Função que executa a busca das paylists do controler **/
public void buscarPlaylist(){
        // Inicializa o handle
    handler = new Handler();
    // Exibe a tela de carregando, ocultando os demais componentes
    exibir(1);
    // Cria Thread para carregar as playlists
    new Thread(new Runnable() {
        public void run() {
                // Executa a rotina do controler para buscar as playlists no servidor
            control.readPlaylist();
            // Ativa o handle para atualizar activity após buscar as playlists
                handler.post(new Runnable() {
                    @Override
                    public void run() {
                        // Executa a função de exibição das playlists
                        showPlaylist();
                    }
                });
        }
    }).start();
}
/** Função para exibir e ocultar itens na tela de acordo com a ação:
     acao = 1 : Exibe apenas o carregando
     acao = 2 : Exibe a lista de playists
     acao = 3 : Exibe mensagem
 * **/
public void exibir(int acao, String Mensagem){
        ListView lv = (ListView) findViewById(R.id.playlist_list);
```

```
        ProgressBar pg = (ProgressBar)findViewById(R.id.progressPlaylist);
        TextView tx = (TextView)findViewById(R.id.playlist_mensagem);

        if(acao == 1){
            lv.setVisibility(View.INVISIBLE);
            tx.setVisibility(View.INVISIBLE);
            pg.setVisibility(View.VISIBLE);
        }else if(acao == 2){
            lv.setVisibility(View.VISIBLE);
            tx.setVisibility(View.INVISIBLE);
            pg.setVisibility(View.INVISIBLE);
        }else if(acao== 3){
            tx.setText(Mensagem);
            lv.setVisibility(View.INVISIBLE);
            pg.setVisibility(View.INVISIBLE);
            tx.setVisibility(View.VISIBLE);
        }
    }
    /** Função de exibir sem mensagem **/
    public void exibir(int acao){
            exibir(acao, "");
    }

    /** Função que exibe a playlist **/
    public void showPlaylist(){
            /** Verifica se existe alguma playlist disponivel **/
            if(control.plays.size()==0){
                    /** Exibe mensagem de erro, caso status_consulta, não houve erro na consulta
e apenas não tem playlists disponiveis **/
                    exibir(3,control.status_consulta ==0 ? "Nenhuma playlist disponível." : "Não
foi possivel conectar ao servidor");
                    return;
            }
        // Gera uma lista de Strings para passar para o ListView da tela do sistema
```

```
    ArrayList<String> your_array_list = new ArrayList<String>();
  for(playlist play: control.plays ){
    your_array_list.add(play.name);
  }

  // Busca objeto listView
    ListView lv = (ListView) findViewById(R.id.playlist_list);

  // Cria Adapter e envia ele para a listView, exibindo a lista de playlist
  ArrayAdapter<String> arrayAdapter =
  new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1, your_array_list);
  lv.setAdapter(arrayAdapter);

  // Cria função de evento para quando algum item da list view for selecionada
  lv.setOnItemClickListener(new OnItemClickListener() {
   public void onItemClick(AdapterView<?> parent, View view,
      int position, long id) {

      // Selecioa a playlist no controle do o id do item selecionado
      control.selectPlayList((int)id);
      MusicList.inicializado = false;
      // Carrega nova activity para exibir a lista de músicas
      Intent intent = new Intent(getApplicationContext(), MusicList.class);

      // Inicia a activiti exibindo a lista de músicas
      startActivity(intent);
    }
  });
  // Exibe a lista de playlists para o usuário
  exibir(2);
}
@Override
protected void onDestroy() {
  super.onDestroy();
```

```
        if(isFinishing()){
            control.encerrarMediaPlayer();
        }
    }
    /** Função responsável por criar o menu da tela de lista de música **/
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.activity_main, menu);
        return true;
    }
    /** Função responsável por controlar a opção selecionada no menu **/
    public boolean onOptionsItemSelected(MenuItem item) {
        // Handle item selection
        switch (item.getItemId()) {
            case R.id.menu_refresh_playlist:
                    buscarPlaylist();
                return true;
            case R.id.menu_settings:
              // Carrega nova activity para exibir a tela de configurações
              Intent intent = new Intent(getApplicationContext(), Configuracoes.class);

              // Inicia a activiti exibindo a lista de músicas
              startActivity(intent);
                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

B.2 - activity_main.xml

```
<!-- XML  com configuração do layout da tela de playlists -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
```

```
    android:layout_width="wrap_content"
    android:layout_height="match_parent" >

    <ProgressBar
        android:id="@+id/progressPlaylist"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true" />
            <ListView
                android:id="@+id/playlist_list"
                android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:visibility="invisible"/>

            <TextView
                android:id="@+id/playlist_mensagem"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerHorizontal="true"
                android:layout_centerVertical="true"
                android:gravity="center_horizontal"
                android:paddingLeft="20dp"
                android:paddingRight="20dp"
                android:textSize="20dp"
        android:visibility="invisible" />
</RelativeLayout>
```

B.3 – MusicList.java

```
/** Classe responsável pelo gerenciamento da tela de exibição da lista de músicas **/
package com.emerson.streaming.client;

import java.util.ArrayList;
```

```
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.view.Menu;
import android.view.MenuInflater;
import android.view.MenuItem;
import android.view.View;
import android.widget.AdapterView;
import android.widget.AdapterView.OnItemClickListener;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.ProgressBar;
import android.widget.TextView;

import com.emerson.streaming.controller.controller;
import com.emerson.streaming.controller.musica;

public class MusicList extends Activity {
    controller control;

    public Handler handler;

    // Flag que indica se a tela de músicas já foi inicializado
    public static boolean inicializado = false;

    @Override
    public void onCreate(Bundle savedInstanceState) {
            //Inicializam componentes graficos
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_music_list);

        control = controller.getInstance();
        setTitle("Lista:" + control.getPlayslistName());
```

```
    // Caso inicializado = false, executa rotina para buscar as músicas
    if(!inicializado)
        buscarMusicas();
        // Caso inicializado = true, executa a rotina que exibe as músicas
    else
        showMusicas();
    inicializado = true;
}
/** Função que executa a busca das músicas do controler **/
public void buscarMusicas(){
        // Inicializa o handle
    handler = new Handler();
    // Exibe a tela de carregando, ocultando os demais componentes
    exibir(1);
    // Cria Thread para carregar as músicas
    new Thread(new Runnable() {
        public void run() {
                // Executa a rotina do controler para buscar as músicas no servidor
                control.readMusicPlaylist();
            // Ativa o handle para atualizar activity após buscar as músicas
                handler.post(new Runnable() {
                    @Override
                    public void run() {
                        // Executa a função de exibição das músicas
                        showMusicas();
                    }
                });
        }
    }).start();
}

/** Função para exibir e ocultar itens na tela de acordo com a ação:
            acao = 1 : Exibe apenas o carregando
```

```
            acao = 2 : Exibe a lista de playists
            acao = 3 : Exibe mensagem
        **/
        public void exibir(int acao, String Mensagem){
                ListView lv = (ListView) findViewById(R.id.music_list);
          ProgressBar pg = (ProgressBar)findViewById(R.id.progressMusic);
          TextView tx = (TextView)findViewById(R.id.music_mensagem);

          if(acao == 1){
            lv.setVisibility(View.INVISIBLE);
            tx.setVisibility(View.INVISIBLE);
            pg.setVisibility(View.VISIBLE);
          }else if(acao == 2){
            lv.setVisibility(View.VISIBLE);
            tx.setVisibility(View.INVISIBLE);
            pg.setVisibility(View.INVISIBLE);
          }else if(acao== 3){
                tx.setText(Mensagem);
            lv.setVisibility(View.INVISIBLE);
            pg.setVisibility(View.INVISIBLE);
            tx.setVisibility(View.VISIBLE);
          }
        }
        /** Função de exibir sem mensagem **/
        public void exibir(int acao){
                exibir(acao, "");
        }
    public void showMusicas(){
      //Pega as playlists
      if(control.getMusicas().size()==0){
                /** Exibe mensagem de erro, caso status_consulta, não houve erro na consulta
e apenas não tem músicas disponiveis **/
                exibir(3,control.status_consulta ==0 ? "Nenhuma músicas disponível." : "Não
foi possivel conectar ao servidor");
```

```
            return;

}
// Cria uma lista de String contendo a descrição que será exibida de cada música
ArrayList<String> your_array_list = new ArrayList<String>();
for(musica musica: control.getMusicas()){
   your_array_list.add(musica.descricao);
}

// Busca objeto listView
   ListView lv = (ListView) findViewById(R.id.music_list);

// Cria Adapter e envia ele para a listView, exibindo a lista de músicas
ArrayAdapter<String> arrayAdapter =
new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1, your_array_list);
lv.setAdapter(arrayAdapter);

// Cria função de evento para quando algum item da list view for selecionada
lv.setOnItemClickListener(new OnItemClickListener() {
  public void onItemClick(AdapterView<?> parent, View view,
     int position, long id) {

     boolean start = !(control.index_musica_atual == id);
     control.selectMusica((int)id);

     // Launching new Activity on selecting single List Item
     Intent i = new Intent(getApplicationContext(), Player.class);
     // sending data to new activity
     startActivity(i);

     if(start)
            control.IniciarReproducao();
  }
});
```

```
        exibir(2);
    }
    /** Função responsável por criar o menu da tela de lista de música **/
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        MenuInflater inflater = getMenuInflater();
        inflater.inflate(R.menu.activity_main, menu);
        return true;
    }
    /** Função responsável por controlar a opção selecionada no menu **/
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            case R.id.menu_refresh_playlist:
                    buscarMusicas();
                return true;
            case R.id.menu_settings:
                    // Carrega nova activity para exibir a tela de configurações
                    Intent intent = new Intent(getApplicationContext(), Configuracoes.class);

                    // Inicia a activiti exibindo a lista de músicas
                    startActivity(intent);

                return true;
            default:
                return super.onOptionsItemSelected(item);
        }
    }
}
```

B.4 - activity_music_list.xml

```
<!—XML de configuração do layout da lista de músicas -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
```

```
    android:layout_height="match_parent" >

    <ProgressBar
        android:id="@+id/progressMusic"
        style="?android:attr/progressBarStyleLarge"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_centerHorizontal="true"
        android:layout_centerVertical="true" />
            <ListView
                android:id="@+id/music_list"
                android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:visibility="invisible"
        >
            </ListView>
            <TextView
                android:id="@+id/music_mensagem"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_centerHorizontal="true"
                android:layout_centerVertical="true"
                android:gravity="center_horizontal"
                android:paddingLeft="20dp"
                android:paddingRight="20dp"
                android:textSize="20dp"
        android:visibility="invisible" />
</RelativeLayout>
```

B.5 – controller.java

```
/** Classe responsável pela gestão das playlists e das músicas, assim como a gestão da
reprodução das músicas, busca das playlists e músicas no servidor **/
package com.emerson.streaming.controller;


import java.util.ArrayList;
```

```
import java.util.Date;

import org.ksoap2.SoapEnvelope;
import org.ksoap2.serialization.SoapObject;
import org.ksoap2.serialization.SoapSerializationEnvelope;
import org.ksoap2.transport.HttpTransportSE;

import android.content.Context;
import android.content.SharedPreferences;
import android.media.MediaPlayer;
import android.media.MediaPlayer.OnBufferingUpdateListener;
import android.media.MediaPlayer.OnCompletionListener;
import android.media.MediaPlayer.OnPreparedListener;
import android.util.Log;
import android.widget.Toast;

import com.emerson.streaming.client.Player;


public class controller {


        private static String NAMESPACE = "http://tempuri.org/";
        private static String GET_PLAYLIST = "getPlaylist";
        private static String SOAP_GET_PLAYLIST = NAMESPACE + "Icommunicate/" +
GET_PLAYLIST;
        private static String GET_MUSICAS = "getMusica";
        private static String SOAP_GET_MUSICAS = NAMESPACE + "Icommunicate/" +
GET_MUSICAS;

        /** Constantes de Configuração **/
        private static final String PREFS_NAME = "STREAMING_CLIENT";
        private static final String CONF_URL_SERVER = "SERVER_URL";
```

```
public static String SERVER_URL;
private static String URL_WEBSERVICE;
private static String URL_STREAMING;

// Instância estatica da classe controler
static controller control;
// Lista de playlits
public ArrayList<playlist> plays;
// número da playlist atual
public int play_list_atual = 0;
// Objeto da playlist atual
public playlist playlist_atual = null;

// numero da música atual
public int index_musica_atual = 0;
public musica musica_atual = null;
// Status da consulta ao servidor ( 0 - Normal ; -1 - Falha )
public int status_consulta = 0;



// Objeto player, responsavel por reproduzir a música
public MediaPlayer player = new MediaPlayer();
public Player PlayerInstante = null;
public int buffer;
boolean tocando = false;

// Objeto Context da Aplicação
public Context context = null;

static public long tempo_inicial;
static public boolean shown_time = false;
/** Função que retorna a instância estática do controller **/
static public controller getInstance(){
        // Verifica se já foi inicializada
```

```
        if(control == null){
                // Inicializa instancia
                control = new controller();
        }
        // retorna a instância do controller
        return control;
}
/** Função de inicialização da classe private **/
private controller(){
        // Inicializa a lista de playlists
        plays = new ArrayList<playlist>();
}
/** Função para selecionar a playlist atual **/
public void selectPlayList(int id){
        try{
                // Tenta setar o id e selecionar a lista atual
                play_list_atual = id;
                playlist_atual = plays.get(id);
                index_musica_atual = -1;
        }catch (Exception e) {
                // Em caso de falha, reseta a lista selecionada
                play_list_atual = 0;
                playlist_atual = null;
        }
}
/** Função que retorna a lista de músicas da playilst atual **/
public ArrayList<musica> getMusicas(){
        // Verifica se existe uma playlist selecionada e retorna sua lista de músicas
        if(playlist_atual != null)
                return playlist_atual.musicas;
        // caso negativo, retorna uma lista de músicas vazia
        else
                return new ArrayList<musica>();
}
```

```
        /** Função que retorna o nome da lista atual **/
        public String getPlayslistName(){
                playlist lista_atual = plays.get(play_list_atual);
                return lista_atual.name;
        }

        public void selectMusica(int index){
                index_musica_atual = index;
                musica_atual = playlist_atual.musicas.get(index);
        }
        /** Função que busca as playlists no servidor **/
        public void readPlaylist(){

                plays = new ArrayList<playlist>();
                status_consulta = 0;

                //Initialize soap request
        SoapObject request = new SoapObject(NAMESPACE, GET_PLAYLIST);

        // envelope da mensagem
        SoapSerializationEnvelope envelope = new
SoapSerializationEnvelope(SoapEnvelope.VER11);
        envelope.setOutputSoapObject(request);
        envelope.dotNet = true;

        // Objeto que envia a requisição
                HttpTransportSE httpTransport = new HttpTransportSE(URL_WEBSERVICE);
        try {
                //this is the actual part that will call the webservice
                httpTransport.call(SOAP_GET_PLAYLIST, envelope);

                        // Get the SoapResult from the envelope body.
                SoapObject result = (SoapObject)envelope.bodyIn;
```

```
                result = (SoapObject)result.getProperty(0);
                int eCode = Integer.parseInt(result.getProperty(0).toString());
                if(eCode == 0){
                        SoapObject lista = (SoapObject)result.getProperty(1);
                        for(int index=0;index< lista.getPropertyCount();index++ ){
                                SoapObject cplaylist = (SoapObject)lista.getProperty(index);
                                plays.add(new
playlist(cplaylist.getProperty("ID").toString(),cplaylist.getProperty("name").toString() ));
                        }
                }
        }catch(Exception error){
                status_consulta = -1;
        }
        }
        /** Função que busca a lista de músicas da playlist atual **/
        public void readMusicPlaylist (){
                //playlist lista_atual = plays.get(play_list_atual);

                status_consulta = 0;
                playlist_atual.clearList();



                //Initialize soap request
        SoapObject request = new SoapObject(NAMESPACE, GET_MUSICAS);
        request.addProperty("PlaylistID", playlist_atual.ID);

        // envelope da mensagem
        SoapSerializationEnvelope envelope = new
SoapSerializationEnvelope(SoapEnvelope.VER11);
        envelope.setOutputSoapObject(request);
        envelope.dotNet = true;

        // Objeto que envia a requisição
                HttpTransportSE httpTransport = new HttpTransportSE(URL_WEBSERVICE);
```

```
                httpTransport.debug = true;

                try {
                //this is the actual part that will call the webservice
                httpTransport.call(SOAP_GET_MUSICAS, envelope);

                        // Get the SoapResult from the envelope body.
                SoapObject result = (SoapObject)envelope.bodyIn;

                result = (SoapObject)result.getProperty(0);
                int eCode = Integer.parseInt(result.getProperty(0).toString());
                if(eCode == 0){
                        SoapObject lista = (SoapObject)result.getProperty(1);
                        for(int index=0;index< lista.getPropertyCount();index++ ){
                                SoapObject cplaylist = (SoapObject)lista.getProperty(index);

                                playlist_atual.AdicionarMusica(new
musica(cplaylist.getProperty("ID").toString()

                , cplaylist.getProperty("titulo").toString()

                   , cplaylist.getProperty("artista").toString()

                   , cplaylist.getProperty("descricao").toString()));
                        }
                }
        } catch (Exception e) {
                status_consulta = -1;
        }
           }
           /** Função responsável por iniciar a reprodução da música **/
           public void IniciarReproducao(){

                try{
```

```
                    // Inicialização da reprodução da música
                    tocando = false;
                    buffer = 0;
                    shown_time = false;
                    tempo_inicial = (new Date()).getTime();
                    player.reset();
                    player.setDataSource(URL_STREAMING + playlist_atual.ID + "/" +
musica_atual.ID);
                    player.prepareAsync();

                    // Evento disparado quando o buffer da música atinge um nivel em que
ela possa ser reproduzida
                    player.setOnPreparedListener(new OnPreparedListener() {
                  public void onPrepared(MediaPlayer mp) {
                    // inicia a reprodução da música
                    mp.start();
                    tocando = true;
                    // Calcula e exibe o tempo que levou para iniciar a músic, em
milisegundos
                    long tempo_final = (new Date()).getTime();
                    Toast.makeText(context,"Tempo de início: " +
Integer.toString((int)(tempo_final - tempo_inicial)), Toast.LENGTH_LONG).show();

                    // Atualiza a tela da música caso ela exista
                    if(PlayerInstante != null){
                            PlayerInstante.AtualizarTela();
                    }

                  }
                });
                    // Evento disparado para atualizar a situação do buffer da música
                    player.setOnBufferingUpdateListener( new
OnBufferingUpdateListener() {
                            @Override
```

```
                    public void onBufferingUpdate(MediaPlayer mp, int percent) {
                        // salva a informação no controller
                        buffer = percent;

                        // Caso a música atinja os 100% e ainda não tenha sido
informado na tela, ele exibe o tempo que levou para carregar toda a música no buffer
                        if(percent==100 && !shown_time){
                    long tempo_final = (new Date()).getTime();
                    Toast.makeText(context,"Tempo de início: " +
Integer.toString((int)(tempo_final - tempo_inicial)), Toast.LENGTH_LONG).show();
                    shown_time = true;
                        }

                        //Atualiza a situação do buffer na tela
                if(PlayerInstante != null){
                    PlayerInstante.updateBuffer();
                }
                    }
                });
                // Evento disparado quando a músicas chega ao fim da reprodução
                player.setOnCompletionListener( new OnCompletionListener() {
                    @Override
                    public void onCompletion(MediaPlayer mp) {
                        // Verifica se a música encerrou e manda para a próxima
música
                        if(tocando){
                            ProximaMusica();
                        }
                    }
                });
            }catch (Exception e) {
            }
        }
        /** Função responsável por avançar para a próxima música **/
```

```
    public void ProximaMusica(){
        if(index_musica_atual+1 < playlist_atual.musicas.size()){
            selectMusica(index_musica_atual+1);
        }else{
            selectMusica(0);
        }
        IniciarReproducao();
    if(PlayerInstante != null){
      PlayerInstante.AtualizarTela();
    }
      }
      /** Função responsável por voltar para a música anterior**/
      public void VoltarMusica(){
        if(index_musica_atual-1 < 0){
            selectMusica(playlist_atual.musicas.size()-1);
        }else{
            selectMusica(index_musica_atual-1);
        }
        IniciarReproducao();
    if(PlayerInstante != null){
      PlayerInstante.AtualizarTela();
    }
      }
      /** Função responsável por avançar a música para uma determinada posição, desde
que essa posição já tenha sido carregada no buffer **/
      public void PlayerGoTo(int seek){
        if(tocando){
            int duracao = control.player.getDuration();
            int limite = (duracao * buffer / 100) - 1000;
            if(seek > limite){
                return;
                //seek = limite < 0 ? 0 : limite;
            }
            player.seekTo(seek);
```

```
                        if(!player.isPlaying())
                                player.start();
                }
        }
        /** Função responsável por alternar a reprodução da música entre pausado e
reproduzindo **/
        public boolean PausarVoltar() {
                if(player.isPlaying()){
                        player.pause();
                }else{
                        player.start();
                }
                return player.isPlaying();
        }
        /** Função responsável por encerrar a reprodução da música **/
        public void encerrarMediaPlayer(){
           if (player != null) {
              if(player.isPlaying()) {
                 player.stop();
              }
              player.release();
           }
        }
        /** Função responsável por buscar as configurações do servidor **/
        public void getConfiguracao(){
     SharedPreferences settings =  context.getSharedPreferences(PREFS_NAME, 0);
     SERVER_URL = settings.getString(CONF_URL_SERVER, "127.0.0.1");
          URL_WEBSERVICE = "http://" + SERVER_URL + ":9000?wsld";
          URL_STREAMING = "http://" + SERVER_URL + ":8080/";
        }
        /** Função responsável por salvar as configurações da url do servidor **/
        public void salvarURL(String Url){
                SharedPreferences settings =  context.getSharedPreferences(PREFS_NAME,
0);
```

```
                SharedPreferences.Editor editor = settings.edit();
                editor.putString(CONF_URL_SERVER, Url);
                editor.commit();
                getConfiguracao();
        }
}
```

B.6 – musica.java

```
/** Classe responsável por armazenar as informações da música **/
package com.emerson.streaming.controller;

public class musica {
   public String titulo;
   public String artista;
   public String descricao;
   public String ID;
   public musica(String ID, String titulo, String artista, String descricao){
        this.ID = ID;
        this.titulo = titulo;
        this.artista = artista;
        this.descricao = descricao;
   }
}
```

B.7 – playslit.java

```
/** classe responsável por armazernar a informação da playlists **/
package com.emerson.streaming.controller;

import java.util.ArrayList;

public class playlist {
        // Lista de músicas
        public ArrayList<musica> musicas;
   // Nome da playlist
```

```
        public String name;
        // ID da playlist
        public String ID;
        /** Inicializa o objeto da playlist **/
        public playlist(String in_ID, String in_name){
                name = in_name;
                ID = in_ID;
                clearList();
        }
        /** Inicializa ou limpa a lista de músicas da playlist **/
        public void clearList(){
                musicas = new ArrayList<musica>();
        }
        /** Função para adicionar músicas na playlist **/
   public void AdicionarMusica(musica musica){
      musicas.add(musica);
   }
}
```

B.8 – Player.java

```
/** Classe responsável pela gestão da tela de reprodução da música **/
package com.emerson.streaming.client;


import java.util.Timer;
import java.util.TimerTask;

import android.app.Activity;
import android.os.Bundle;
import android.os.Handler;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.SeekBar;
import android.widget.TextView;
```

```
import com.emerson.streaming.controller.Utilitario;
import com.emerson.streaming.controller.controller;

public class Player extends Activity {
   controller control;
   public Handler handler;
   Thread seeker;
   Button play;
   Button avancar;
   Button voltar;
   TextView artista;
   TextView musica;
   TextView player_duracao;
   TextView player_buffer;
   TextView player_seet_time;
   SeekBar seek;

   Timer timer = new Timer();
   @Override
   public void onCreate(Bundle savedInstanceState) {
         //Inicializam componentes graficos
      super.onCreate(savedInstanceState);
      setContentView(R.layout.player);

      // Configura handle e controler do sistema
         handler = new Handler();
         control = controller.getInstance();
         // Seta a view atual no controler, necessária para resposta do controler a tela de
visualização
         control.PlayerInstante = this;

         // Executa função que configura a tela para o usuário
      AtualizarTela();
```

```
    }
    public void AtualizarTela(){
        try {
            // Caso as váriaveis da tela não tenham sido configuradas
        if(play == null){

            // Busca todos os componentes configuraveis da tela de reprodução
            play = (Button) findViewById(R.id.player_reproduzir);
            avancar= (Button) findViewById(R.id.player_avancar);
            voltar = (Button) findViewById(R.id.player_voltar);
            artista = (TextView) findViewById(R.id.player_artista);
            musica  = (TextView) findViewById(R.id.player_musica);
            player_buffer  = (TextView) findViewById(R.id.player_buffer);
            player_duracao = (TextView) findViewById(R.id.player_duracao);
            player_seet_time = (TextView) findViewById(R.id.player_seek_time);
            seek = (SeekBar) findViewById(R.id.player_seek);

            // Seta evento que será executado toda vez que a posição da barra de
reprodução for alterada manualmente
            seek.setOnSeekBarChangeListener(new SeekBar.OnSeekBarChangeListener()
{
                                int progress;
                                public void onStartTrackingTouch(SeekBar seekBar) {}
                                // Evento disparado quando o usuário tira o dedo da barra
de repodução
                                public void onStopTrackingTouch(SeekBar seekBar) {
                                    // Envia um comando para o controler colocar a
música na posição da barra
                                    control.PlayerGoTo(progress);
                                    // Atualiza a tela do usuáro
                                    AtualizarTela();
                                    // Reinicia o controle da barra de reprodução
                                    startSeek();
```

```
                                    }
                                    // Evento disparado quando a posição da barra de
reprodução é alterada
                                    @Override
                                    public void onProgressChanged(SeekBar seekBar, int
progress,
                                            boolean fromUser) {
                                        // Verifica se a barra foi alterada pelo usuário
                                        if(fromUser){
                                            // Atualiza o progresso com a ultima
posição colocada
                                            this.progress = progress;
                                        }
                                    }
                                });
                // Evento disparado ao clicar no botão play/pause, para pausar ou voltar a
reproduzir a música
                play.setOnClickListener( new OnClickListener() {

                                    @Override
                                    public void onClick(View arg0) {
                                        // Executa a função PausarVoltar, e altera o texto
do botão para representar a ação disponível (Pausa ou reproduzir)
                                        play.setText(control.PausarVoltar() ?
R.string.pause : R.string.play);
                                    }
                                });
                // Evento disparado ao clicar no botão ">>" Avançar
                avancar.setOnClickListener( new OnClickListener() {
                                    @Override
                                    public void onClick(View v) {
                                        // Avança para a proxima música
                                        control.ProximaMusica();
                                    }
```

```
                        });
                // Evento disparado ao clicar no botão "<<" Voltar
                voltar.setOnClickListener(new OnClickListener() {
                                @Override
                                public void onClick(View v) {
                                        // Volta para a música anterior
                                        control.VoltarMusica();
                                }
                        });

        }
        // Seta o nome do artista e da música
        artista.setText(control.musica_atual.artista);
        musica.setText(control.musica_atual.titulo);

        // Seta a duração total da música e configura o valor máximo da barra de reprodução
        if((control.player.getDuration() / (1000*60)) > 200){
                seek.setMax(control.player.getDuration());
                player_duracao.setText(Utilitario.getTimeText(0));
        }
        // Verifica se a música está em execução para configurar o botão de reprodução e
iniciar o controle da barra de reprodução
        if(control.player.isPlaying()){
                play.setText("Pausar");
                startSeek();
        }else{
                play.setText("Reproduzir");
        }
        // Atualiza as informação de carregamento do buffer
        updateBuffer();
        }catch (Exception e) {
                        // TODO: handle exception
                }
    }
```

```
   /** Função responsável por exibir as informações de situação do buffer da música **/
   public void updateBuffer(){

        // Pega a porcentagem atual do buffer
        int percent = control.buffer;
        // Seta a barra secundaria do controle de reprodução, indicando na barra, até que ponto
a música foi carregada
        seek.setSecondaryProgress(control.player.getDuration() * percent / 100);
        // Caso o buffer tenha atingido o 100%, oculta a informação do buffer, se não exibe a
informação do buffer
        player_buffer.setVisibility(percent == 100? View.INVISIBLE : View.VISIBLE);
        // Seta o texto da porcentagem carregada
        player_buffer.setText(Integer.toString(percent) + '%');
   }
   /** Função responsável por atualizar a posição da música na barra de reprodução e o tempo
atual **/
   public void updateSeek(){
        try {
                // atualiza a barra de reprodução
                seek.setProgress(control.player.getCurrentPosition());
                // Atualiza o tempo atual da música

        player_seet_time.setText(Utilitario.getTimeText(control.player.getCurrentPosition()));
        } catch (Exception e) {}
   }
   /** Função responsável por ativar o gerenciamento da posição atual da música**/
        public void startSeek (){
                /** Seta na barra de reprodução, sua posição final, com a duração total da
música **/
                seek.setMax(control.player.getDuration());
                // Seta o texto da duração total da reprodução
        player_duracao.setText(Utilitario.getTimeText(control.player.getDuration()));
        // Inicia agendamento da atualização das informações da reprodução da música,
executada a cada segundo
```

```
            timer.schedule(new TimerTask() {
            public void run() {
                    try {
                   handler.post(new Runnable() {
                     public void run() {
                             // Atualiza as informações da posição atual da música
                             updateSeek();
                     }
                   });
                     }catch (Exception e) {
                                             // Cancela o timer de agendamento, caso algum erro
aconteça na atualização
                             timer.cancel();
                                     }
            }
        }, 1, 1000);
    }
}
```

B.9 - player.xml

```
<!—XML de configuração de layout da tela de reprodução da música -->
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent" >

    <Button
        android:id="@+id/player_voltar"
        style="?android:attr/buttonStyleSmall"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
```

```
    android:text="@string/voltar" />

<Button
    android:id="@+id/player_reproduzir"
    style="?android:attr/buttonStyleSmall"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerHorizontal="true"
    android:text="@string/play" />

<Button
    android:id="@+id/player_avancar"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_alignParentRight="true"
    android:text="@string/avancar" />

<SeekBar
    android:id="@+id/player_seek"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_above="@+id/player_voltar"
    android:layout_alignParentLeft="true" />

<TextView
    android:id="@+id/player_buffer"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:layout_above="@+id/player_seek"
    android:layout_alignParentLeft="true"
    android:gravity="center"
    android:text="0%"
```

```
    android:textAppearance="?android:attr/textAppearanceMedium" />

<TextView
    android:id="@+id/textView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_above="@+id/player_buffer"
    android:layout_centerHorizontal="true"
    android:text="/"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<TextView
    android:id="@+id/player_seek_time"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textView2"
    android:layout_alignBottom="@+id/textView2"
    android:layout_toLeftOf="@+id/textView2"
    android:text="00:00"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<TextView
    android:id="@+id/player_duracao"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_alignBaseline="@+id/textView2"
    android:layout_alignBottom="@+id/textView2"
    android:layout_toRightOf="@+id/textView2"
    android:text="00:00"
    android:textAppearance="?android:attr/textAppearanceMedium" />

<TextView
    android:id="@+id/player_artista"
    android:layout_width="fill_parent"
```

```
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:gravity="center"
        android:text="Artista"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <TextView
        android:id="@+id/player_musica"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/player_artista"
        android:gravity="center"
        android:text="Música"
        android:textAppearance="?android:attr/textAppearanceLarge" />

</RelativeLayout>
```

B.10 – Configuracoes.java

```
/** Classe responsável pela gestão da tela de configuração do sistema **/
package com.emerson.streaming.client;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.EditText;

import com.emerson.streaming.controller.controller;

public class Configuracoes extends Activity {
        // Instância do controler principal
```

```
    controller control = controller.getInstance();

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_configuracoes);

        // Seta o endereço salvo do servidor
        final EditText enderecoServidor = (EditText) findViewById(R.id.endereco_servidor);
        enderecoServidor.setText(control.SERVER_URL);

        // Seta a função de salvar do botão
        Button salvar = (Button) findViewById(R.id.salvar);
        salvar.setOnClickListener(new OnClickListener() {
                        @Override
                        public void onClick(View v) {
                                // Salva a  configuração e fecha a tela de configuração
                                control.salvarURL(enderecoServidor.getText().toString());
                                finish();
                        }
                });
    }
}
```

B.11 - activity_configuracoes.xml

```
<!-- XML de configuração do layout da tela de configuração -->
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent" >

    <TextView
        android:id="@+id/textView1"
        android:layout_width="wrap_content"
```

```
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_alignParentTop="true"
        android:layout_marginTop="17dp"
        android:text="@string/endereco_servidor"
        android:textAppearance="?android:attr/textAppearanceLarge" />

    <EditText
        android:id="@+id/endereco_servidor"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentLeft="true"
        android:layout_below="@+id/textView1"
        android:ems="10" >

        <requestFocus />
    </EditText>

    <Button
        android:id="@+id/salvar"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_alignParentBottom="true"
        android:layout_alignParentLeft="true"
        android:text="@string/salvar" />

</RelativeLayout>
```