FOUD: Um Framework Oportunístico e Ciente de Contexto para Upload de Dados Corporativos

Valdiney Soares de Araújo, Thais Regina de Moura Braga Silva
Instituto de Ciências Exatas - Universidade Federal de Viçosa - Campus Florestal (UFV)
{valdiney.araujo, thais.braga}@ufv.br

RESUMO

Diversas organizações precisam coletar dados onde o acesso a Internet é comprometido. Perícias, levantamentos topográficos, minerações, reportagens, estudos biologicos e fiscalizações ambientais são exemplos destas atividades, que na maioria das vezes são realizadas em zonas rurais, onde a tecnologia de rede sem fio não funciona ou funciona com alcance reduzido. Dessa maneira, os dados obtidos, e que normalmente precisam ser enviados para um servidor central, podem chegar com atraso e/ou erros. Este trabalho apresenta uma alternativa que visa facilitar o serviço de empresas ou orgãos governamentais que trabalham com coleta de dados nestas condições. Trata-se de um *framework*, oportunístico e ciente de contexto, voltado para a construção de sistemas para *upload* de dados coletados em áreas onde o acesso à tecnologia de rede é de baixa qualidade ou inexistente. O modelo produzido para o *framework* realiza o envio de dados quando verifica disponibilidade de rede, priorizando o uso da rede em situações em que energia e pacote de dados estão disponíveis. A implementação desse modelo para o Android mostrou que é possível desenvolvê-lo para um sistema concreto em particular, e que o mesmo pode contemplar particularidades de cada sistema, como os contextos, metadados dos arquivos e lógicas de transmissão mais adequados em cada caso.

Palavras-Chave

Upload de Dados Móveis, Transmissão Oportunística, Sensibilidade ao Contexto

ABSTRACT

Many organizations need to collect data where Internet access is compromised. Skills, surveying, mining, reporting, biological studies, and environmental monitoring are examples of these activities, most of which are carried out in rural areas where wireless network technology does not work or operate at a reduced range. In this way, the data obtained, and which normally need to be sent to a central server, may arrive late and / or error. This paper presents an alternative that aims to facilitate the service of companies or government agencies that work with data collection under these conditions. It is an framework, opportunistic and context-aware, aimed at constructing systems for upload data collected in areas where access to network technology is of low quality or non-existent. The model produced for the framework performs the sending of data when verifying network availability, prioritizing the use of the network in situations in which energy and data packet are available. The implementation of this model for Android showed that it is possible to develop it for a particular system in particular, and that it can contemplate particularities of each system, such as the contexts, meta data of the files and more appropriate transmission logics in each case.

SBSI 2019 Maio 20 – 24, 2019, Aracaju, sergipe, Brazil

Keywords

Mobile Data Upload, Opportunistic Transmission, Context-awareness

1. INTRODUÇÃO

É indiscutível o impacto da mobilidade no cenário tecnológico atual, impulsionando o desenvolvimento de dispositivos móveis e seus aplicativos, bem como de interfaces para comunicação sem fio. As pessoas estão preferindo pagar mais e até, algumas vezes, abrir mão de requisitos importantes, como a segurança da informação e a privacidade do usuário, a ficarem presos a um computador de mesa. No entanto, a instalação destas tecnologias ainda possuem alto custo e não estão disponíveis em todos os locais. Em áreas rurais por exemplo, nem sempre encontramos interfaces de rede de comunicação sem fio que garanta conexão com a Internet de boa qualidade.

Em particular, funcionários públicos e corporativos que precisam frequentemente coletar dados em áreas onde a conexão sem fios com a Internet é ruim ou inexistente, tais como as zonas rurais, acabam não usufruindo de todo potencial das tecnologias móveis. Profissionais como topógrafos, biólogos, peritos, repórteres, entre outros, ainda hoje precisam anotar dados coletados em papéis ou guardar suas informações em aplicativos *offline* para posteriormente fazerem o *upload* dos mesmos em um servidor. Este é um processo dispendicioso e propenso a erros pois há o risco de perda das anotações ou esquecimento de quais dados devem ser enviados.

Geralmente, esses tipos de serviços são feitos por usuários que migram por diversas regiões em um mesmo turno

de trabalho. Durante estes trajetos, é provavel que estas pessoas passem temporariamente por áreas de cobertura de Wifi e/ou 3G que poderiam ser aproveitadas para transmissão parcial e/ou total dos dados. Em outras palavras, os profissionais têm a oportunidade de transmitir seus dados, mas muitas vezes essa oportunidade é desperdiçada. Surge assim a idéia de construir o FOUD - *Framework* Oportunístico para *Upload* de Dados Móveis, que facilite a construção de aplicativos que queiram aproveitar a presença de uma interface de rede sem fio, mesmo que temporária, monitorando o ambiente no qual o dispositivo esteja inserido, principalmente no tocante à características da própria conexão, para realizar a transmissão da melhor forma possível, em termos de diferentes critérios, tais como confiabilidade, segurança, consumo energético do dispositivo, dentre outros.

Para tanto, o *framework* FOUD foi desenvolvido de maneira a ser oportunístico e ciente de contexto. Uma atividade interativa pode ser considerada oportunística quando antecipada por alguém e acontece por acaso [8]. O *framework* é oportunístico porque monitora as interfaces de rede do dispositivo móvel utilizado, verificando a disponibilidade de conexão. Detectada a disponibilidade ele estabelece conexão com o servidor e a usa para fazer *upload*. De acordo com [6], o contexto funciona como um conjunto de restrições que influenciam o comportamento de um sistema. O FOUD é ciente de contexto porque é capaz de avaliar diferentes parâmetros, como por exemplo aqueles relacionados à transmissão, de forma contínua. Informações sobre, por exemplo, custo de uso e qualidade histórica da conexão podem ser utilizadas para decidir quando e como os dados serão transmitidos. O contexto é utilizado para informar ao aplicativo que há chance real de sucesso no *upload* de dados, e portanto, deve iniciá-lo.

Um aspecto importante do *framework* FOUD é a sua extensibilidade, sendo adaptável às necessidades de cada sistema. O desenvolvedor, ao usar o FOUD na implementação do aplicativo de seu sistema, poderá escolher as tecnologias de transmissão a serem utilizadas, bem como a lógica para efetuar o envio dos dados. No que diz respeito ao uso de contextos, o FOUD permite a personalização que os desenvolvedores precisam.

O *framework* FOUD para *upload* de dados corporativos pressupõe que os aplicativos que o utilizarão são formados por uma parte móvel e um servidor. A parte móvel contém o aplicativo, implementado com a utilização do *framework*, que roda em dispositivos como *smartphones* e *tablets*. Através dela, os usuários do sistema coletam e enviam seus dados. O servidor, localizado em nuvem, recebe e armazena estes dados.

Este artigo está organizado da seguinte forma: A Seção 2 possui os trabalhos relacionados encontrados. A Seção 3 apresenta o modelo genérico do *framework*. Uma implementação do FOUD para o Android pode ser encontrada na Seção 4. Na seção 5 temos um estudo de caso. Finalmente, a Seção 6 descreve as considerações finais e os trabalhos futuros.

2. TRABALHOS RELACIONADOS

O trabalho apresentado neste artigo lida com a questão de efetuar *upload* de arquivos de diversos tipos em um servidor remoto, em uma situação de presença intermitente de conexão à Internet. Entretanto, o foco está na apresentação de um *framework* que, ao ser utilizado no desenvolvimento dos aplicativos que farão a geração/coleta dos arquivos, possibilite, de forma automática, a monitoração dos períodos de presença de conexão e envio dos arquivos nestes momentos, utilizando a disponibilidade de rede da forma mais adequada possível, de acordo com os interesses do aplicativo e as características dos dados, do dispositivo móvel e da comunicação em utilização.

Não foram encontrados na literatura trabalhos cujo objetivo seja o mesmo do abordado neste artigo. Por outro lado, pode-se considerar como relacionados, trabalhos que lidam de alguma maneira com o *upload* de dados móveis, porém com o foco diferente.

Os trabalhos [10, 5, 7] exploram a questão da infraestrutura do lado do servidor ou ainda da nuvem para a qual será feita o *upload* de dados coletados por dispositivos móveis. Neles, questões referentes à configuração, escolha e conexão sem fio à nuvem de armazenamento de dados são tratadas, bem como procura-se abordar a melhor forma de se realizar o armazenamento em si dos dados enviados.

Nosso trabalho difere-se destes devido ao fato de que o *framework* FOUD foi inteiramente desenvolvido para ser utilizado pelo aplicativo que é executado no dispositivo móvel e que fará o envio dos dados. Ele considera, como premissa, que o servidor ou nuvem exista e que funcione corretamente, sem contudo lidar com as questões de melhoria de acesso ou funcionamento interno deste.

Em [9] pode ser encontrado um trabalho ligado à área de Redes Tolerantes à Atrasos e Interrupções (*Delay Tolerant Networks - DTNs*), cujo foco está na proposição de abordagens ligadas à infraestrutura da rede. Neste tipo de artigo, pressupõe-se que os dispositivos móveis atravessarão períodos de desconexão, e que, portanto, são necessárias estratégias internas à própria rede que possam garantir, tanto quanto possível, a entrega dos dados a serem comunicados. O dispositivo móvel e suas aplicações não são envolvidos nas estratégias que visam contornar os períodos de interrupção de acesso, e sim os elementos internos à rede e seus protocolos. As DTNs, inclusive, consideram a desconexão não apenas do dispositivo móvel que deseja realizar a transmissão de dados, mas de todos os elementos que compõem a rede de comunicação. O *framework* FOUD, por sua vez, não trata da possibilidade de desconexão interna na rede de comunicação. Ele apenas considera que o dispositivo móvel passará por períodos de conexão intermitente e realiza um trabalho para utilizar os períodos de acesso da melhor forma possível.

Podemos ainda ressaltar que já existem ferramentas disponíveis no mercado para a realização de *download* e *upload* de dados armazenados em nuvem, as quais podem ser utilizadas a partir de diversos tipos de equipamentos computacionais, inclusive de dispositivos móveis. Exemplos de ferramentas como essas são o DropBox [1], GoogleDrive [2] e o OneDrive [3]. As principais diferenças entre essas ferramentas, e a proposta apresentada pelo *framework* FOUD, residem no fato de que as mesmas não mantém o foco apenas no *upload*, além de precisarem lidar com a questão de sincronização de arquivos. Tais soluções não se configuram como um *framework* e não disponibilizam APIs, de maneira que possam ser utilizadas de forma integrada no desenvolvimento de aplicativos de coleta de dados/arquivos.

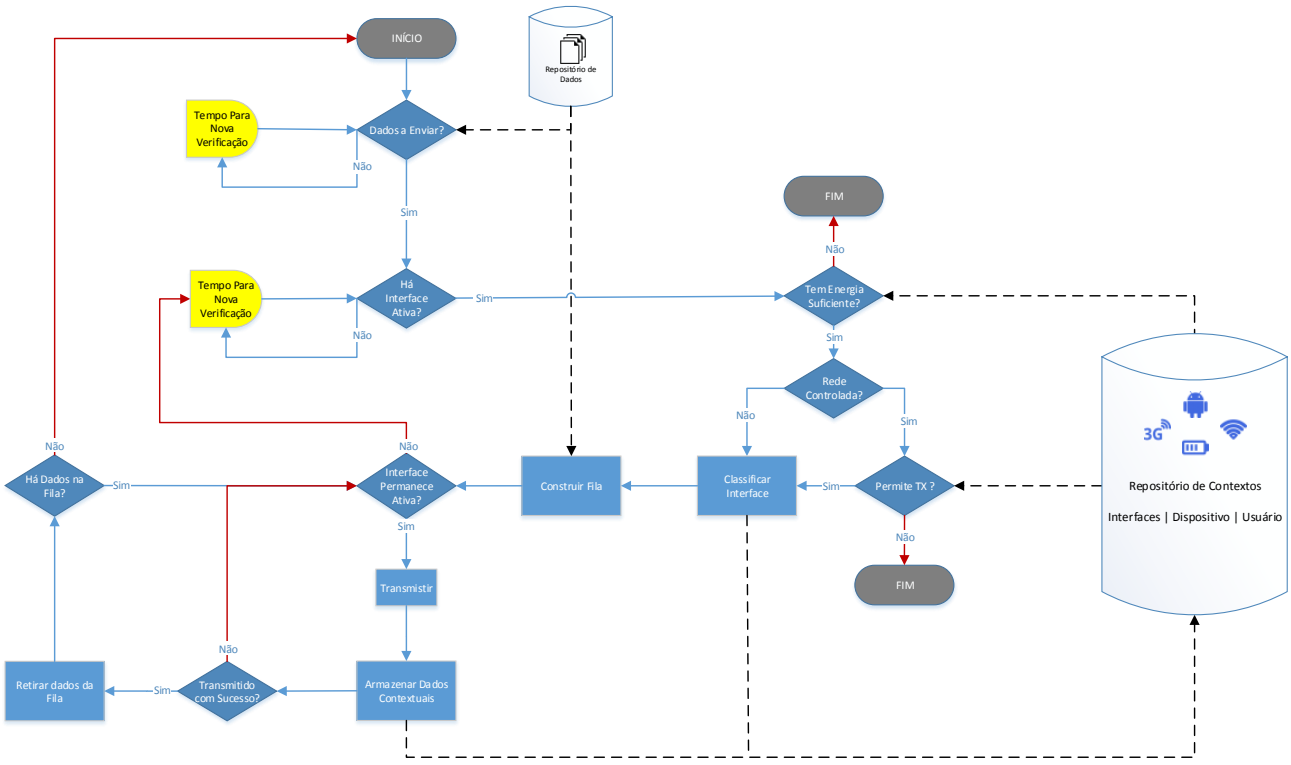

Figura 1: Roteiro para funcionamento do FOUD

3. O MODELO DO *FRAMEWORK* FOUD

O FOUD trabalha com a ideia de que o usuário do aplicativo móvel deve gerar diversos tipos de arquivos de dados, os quais serão colocados em um diretório padrão para transmissão oportunística. O *framework* então, verificando a quantidade de energia no dispositivo e presença de conexão de rede, constrói uma fila de transmissão com os arquivos que estão no diretório, utilizando para tanto, critérios configuráveis de prioridade. As informações utilizadas para a construção dessa fila são obtidas de metadados dos arquivos a serem transmitidos, bem como de repositórios de dados contextuais mantidos pelo FOUD. Por fim, os arquivos vão sendo transmitidos, um a um, de acordo com a ordem da fila, havendo chance de retransmissão em caso de falha, mas com preservação da conexão. Arquivos transmitidos com sucesso são removidos da fila. A fila só é reconstruída, caso o fluxo do *framework* retorne ao seu ponto inicial, sendo que isso acontece quando a conexão é perdida ou quando todos os *uploads* previstos na fila são executados.

O *framework* atua sobre três pilares principais: repositório de arquivos, repositório de contexto e núcleo de transmissão oportunística. No roteiro ilustrado na figura 1, podemos ver o repositório de arquivos e de contextos através de dois símbolos representativos de base de dados. O restante do roteiro é o que denominamos núcleo de transmissão oportunística.

O repositório de arquivos constitui uma estrutura criada no *framework* que permite o monitoramento de arquivos dispostos em diretórios pré-determinados. O *framework* FOUD cria um diretório no dispositivo onde o aplicativo é instalado. Neste diretório são criadas três pastas: "A enviar", "Enviando" e "Enviados". A pasta "A enviar" é onde o aplicativo construído deve dispor seus arquivos para *upload*. Enquanto as condições iniciais de funcionamento do *framework* não forem satisfeitas, estes arquivos permanecem nesta pasta. Satisfeitas estas condições iniciais, os arquivos são transferidos para a pasta "Enviando". Cada arquivo transmitido é movido para a pasta "Enviados".

O repositório de contextos é onde se encontra o conjunto de informações utilizadas para tomada de decisão do *framework* quanto à transmissão de arquivos. Ele é constituído por duas partes: um banco de dados e por estruturas de dados a serem criadas pelo desenvolvedor.

O banco de dados é utilizado para guardar informações sobre transmissões de arquivos, como identificação da rede, tamanho do arquivo transmitido e tempo de transmissão.

Quanto às estruturas de dados a serem criadas pelo desenvolvedor, estas servem para o armazenamento de variáveis que possuem alguma relação com a transmissão de arquivos. Geralmente, estas variáveis estarão vinculadas ao dispositivo, ao usuário ou à própria interface de rede. Atualmente um dispositivo possui inúmeros sensores que podem ser monitorados com vista a auxiliar no processo de upload de dados. São sensores de movimentos (aceleração e rotação), ambientais (temperatura, pressão, iluminação e umidade) e até de posição (orientação), que podem apresentar alta correlação com a qualidade de transmissão de arquivos ou até mais que isso, uma relação de causalidade. Mesmos que estes elementos não estejam relacionados diretamente com a qua-

lidade de *uploads*, eles podem ser importantes para permitir aos desenvolvedores uma marcação para transmissão de seus arquivos. Um calendário, por exemplo, poderia ser utilizados para transmitir um arquivo em determinada data. Este *framework* não traz restrições quando ao número de sensores ou variáveis a serem utilizados. Qualquer sensor existente, acoplado de forma independente ou inserido nativamente em futuros dispositivos, poderá ser monitorado. Quanto maior o número de sensores, mais rico é o contexto no qual o *framework* se embasa para tomada de decisão no que se refere à transmissão dos arquivos.

O núcleo de transmissão oportunística, último pilar citado, é a parte central do *framework*. Este controlador geral interliga, de forma possivelmente desacoplada, todos os elementos genéricos necessários para uma transmissão eficiente de dados e se resume em verificar as restrições de funcionamento do FOUD, ordenar os arquivos monitorados pelo repositório de arquivos e transmiti-los. Acompanhando o roteiro ilustrado na figura 1, obteremos maiores detalhes.

As três primeiras etapas, "**Dados a Enviar?**", "**Há Interface Ativa?**" e "**Tem Energia Suficiente?** constituem as restrições básicas para o seu funcionamento. Isso quer dizer que o funcionamento do FOUD está condicionado à existência de arquivos e que nenhum arquivo encontrado no diretório padrão será transmitido caso não haja rede disponível e uma quantidade de energia preestabelecida que mantenha o dispositivo móvel em funcionamento após a realização do *upload*. Entendemos por quantidade de energia preestabelecida, uma porcentagem especificada pelo desenvolvedor, que indique o nível mínimo permitido para início do procedimento de transmissão.

Atendidas estas restrições, o próximo passo a ser considerado é verificar se a interface de rede a qual o dispositivo se conectou possui algum tio de controle no que se refere ao uso de pacotes de dados adquiridos da operadora - etapa "**Interface Controlada?**". Podemos ter uma interface de rede sem controle sobre uso de pacotes, ou com controle, na qual o uso deste pacote de dados é limitado ou proibido pelo usuário final. Tratando-se de interface de rede controlada na qual é proibido o uso de pacote de dados, o FOUD não tenta utilizar esta rede para transmissão de arquivos.

Uma vez que a interface de rede encontrada permita, a princípio, a transmissão dos dados, é boa a hora para nos preocuparmos com as informações preliminares a seu respeito. Na etapa deste roteiro denominada "**Classificar Interface**", estas informações serão recuperadas do banco de dados do repositório de contextos, quando houverem, e são consideradas essenciais para a etapa posterior, "**Construir Fila**", quando priorizaremos o posicionamento dos arquivos. O desenvolvedor deverá construir algoritmos para ordenação dos arquivos e lógicas para escolha destes algoritmos em tempo de execução. Portanto, cada mudança nos atributos do contexto monitorado poderá culminar na execução de um algoritmo de ordenação diferente.

A próxima etapa, "**Interface Permanece Ativa?**", se faz necessária devido a movimentação constante do dispositivo. Mesmo sob abrangência de uma mesma interface de rede, esta movimentação pode trazer experiências diversas relacionadas a qualidade de transmissão. Somente após confirmação das condições de conexão é que se pode progredir e passar para próxima etapa: "**Transmitir**".

Na etapa de transmissão, o arquivo pode ter sido transmitido com sucesso ou não. Após a etapa de transmissão, o FOUD executará a tarefa de **"Armazenar Dados Contextuais"**. Esta fase constitui em persistir dados referentes à transmissão. Na etapa posterior, **"Transmitido com Sucesso?"**, verifica-se como foi a transmissão do arquivo. No caso de insucesso no *upload* e confirmada a incapacidade de envio do arquivo por esta interface de rede, retonar-se para a etapa inicial, onde o FOUD fica aguardando nova conexão. No caso de sucesso na transmissão, o arquivo transmitido é retirado da fila - etapa **"Retirar Dados da Fila"** e na sequência é verificada a existência de outros arquivos a serem transmitidos - etapa **"Há Dados na Fila?"**. Confirmada a existência de mais arquivos, um novo arquivo assume posição para próxima transmissão. Caso contrário, o ciclo é reiniciado.

4. IMPLEMENTAÇÃO DO FOUD PARA O SISTEMA OPERACIONAL ANDROID

4.1 Uso da biblioteca *WorkManager*

Desde o Android Marshmallow, uma série de mudanças estão sendo realizadas a fim de garantir a segurança, economia de energia e memória dos dispositivos. As versões recentes do Android tornam estas limitações muito mais rigorosas, principalmente no que se refere a execução de serviços em segundo plano.

A execução de tarefas em background no Android leva em consideração um amontoado de fatores. Se o aplicativo estiver em execução, estas tarefas podem ser realizadas no mesmo processo através de Threads, senão, outras formas mais aprimoradas deverão ser utilizadas e a escolha de cada uma destas formas ainda dependerá da versão do Android (nível da API). O WorkManager condensa estas possibilidades e permite restringir a execução de tarefas à certas condições estabelecidas pelo desenvolvedor, como interface de rede ativa, por exemplo.

O WorkManager é destinado a tarefas que exigem uma garantia de que o sistema as executará mesmo se o aplicativo for encerrado, como o upload de dados do aplicativo para um servidor [4]. A biblioteca permite ainda, agendar tarefas individuais em um intervalo especificado, encadear sequências de tarefas ou cancelá-las. O seu uso no framework FOUD garante sua execução em background e reduz a sua complexidade.

4.2 Diagrama de Classes

O *framework* foi implementado conforme apresentado no diagrama de classes da figura 2. Este diagrama obedece o roteiro apresentado na seção anterior e nele também podemos observar os três pilares do *framework* FOUD: A classe RepositorioDeDados, a classe EstatisticaDAO, que representa parte do repositório de contextos e, por fim, a classe NucleoDeTransmissaoOportunistica. A fim de facilitar a leitura, a descrição do papel de cada uma destas classes será realizada de cima para baixo no diagrama.

Neste diagrama temos a classe **Foud**, uma classe de fronteira que se relaciona com outras de forma semelhante ao padrão de projeto *Façade*. Ao instanciar esta classe o desenvolvedor deve passar como parâmetro uma classe que estende da classe NucleoDeTransmissaoOportunistica. A classe Foud é responsável por chamar métodos estáticos das classes CriadorDiretorio e GerenciadorFoud.

A classe **CriadorDiretorio** possui um método estático denominado criarDiretorio( ), que cria o diretório principal

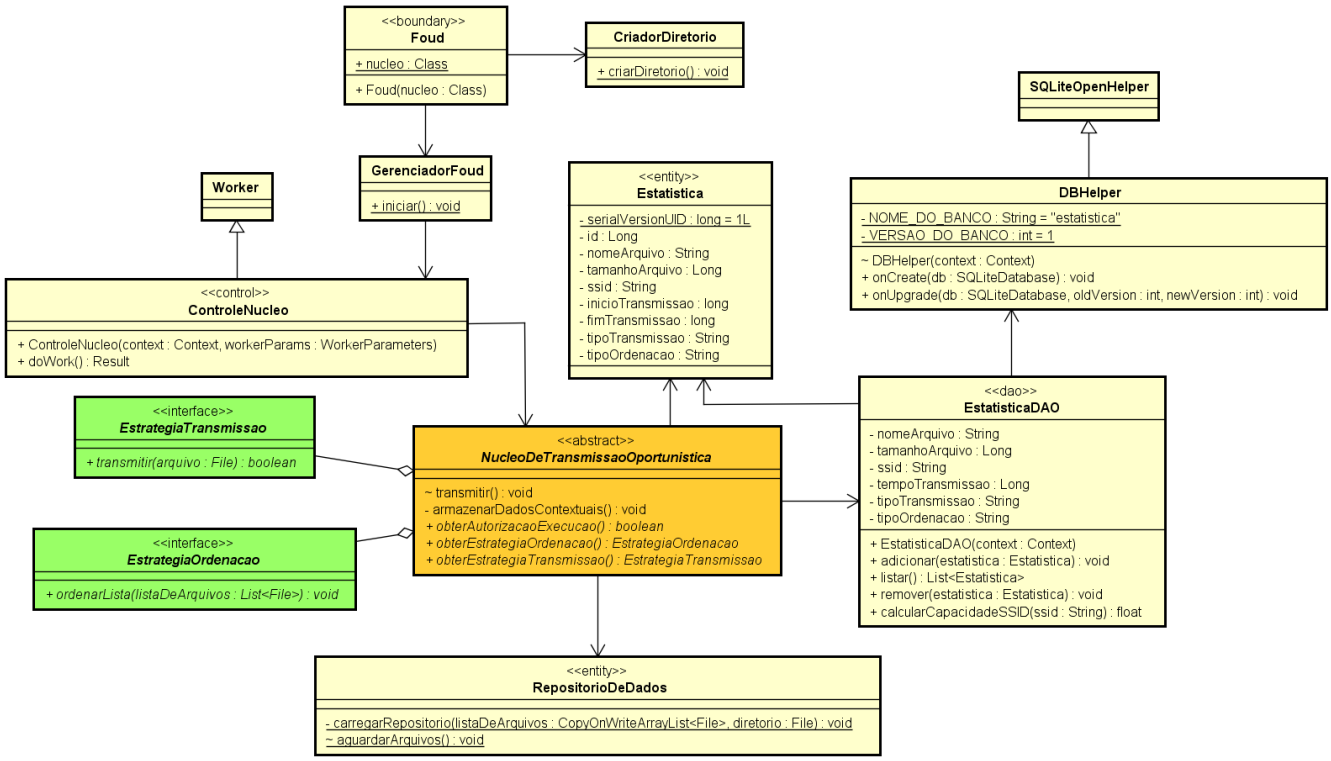

Figura 2: Diagrama de Classes FOUD

do *framework* no dispositivo e três subdiretórios: "A enviar", "Enviando" e "Enviados".

A classe **GerenciadorFoud** possui um método estático denominado iniciar( ), que aciona o ControleNucleo, um controlador do sistema.

A classe **ControleNucleo** estende da classe ***Worker***. Esta é uma condicionante ao utilizar a biblioteca **WorkManager**. Ao estender a classe *Worker*, temos de sobrescrever o método *doWork( )*, onde é realizado todo trabalho em segundo plano. No nosso *framework* este trabalho se resume em chamar o método transmitir( ) da classe NucleoDeTransmissaoOportunistica.

A classe central do *framework* é denominada **NucleoDeTransmissaoOportunistica**. Trata-se de uma classe abstrata que se encaixa no padrão de projetos *Template Method*. O desenvolvedor deve estender esta classe e sobrescrever os seus métodos. É através destes métodos que podemos criar condições lógicas para execução do *framework* e alteração das estratégias de transmissão e de ordenação. O principal objetivo desta classe é transmissão de arquivos. Resumidamente sua atuação é verificar a existência de arquivos junto a classe RepositorioDeDados, ordená-los e transmiti-los. Informações referentes a cada transmissão de arquivo são persistidas em um banco de dados através da classe EstatisticaDAO.

A classe **RepositorioDeDados** é responsável por manter uma lista temporária de todos arquivos a serem transmitidos. Ela monitora o diretório do *framework* a fim de manter estas listas atualizadas.

A classe **EstatisticaDAO** é utilizada para persistir informações como nome e tamanho do arquivo, identificação da rede e tempo de transmissão. O banco de dados utilizado é o *SQLite*.

Neste diagrama temos também duas interfaces: **EstrategiaTransmissao** e **EstrategiaOrdenacao**. Estas interfaces, agregadas a classe NucleoDeTransmissaoOportunistica, atuam conforme os preceitos do padrão de projeto *Strategy*. Desta forma, podemos implementar cada uma das interfaces quantas vezes forem necessárias. Estas estratégias podem ser alternadas em tempo de execução, de acordo com as lógicas criadas pelo desenvolvedor e os contextos em que o dispositivo estiver inserido.

5. ESTUDO DE CASO

Para utilizar o FOUD, o desenvolvedor deve importar a respectiva biblioteca e seguir os seguintes passos:

1. Criar classe(s) de contexto.

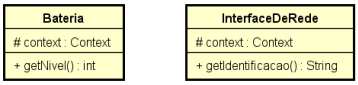

Figura 3: Criação de classes de contexto.

As classes de contexto (figura 3) devem ser criadas pelo desenvolvedor. Estas classes são as responsáveis

por armazenar variáveis que possuem alguma correlação com a transmissão de arquivos. Para efeito deste estudo de caso, criamos duas classes de contexto: Bateria e InterfaceDeRede.

2. Criar classe(s) que implementem a interface “EstrategiaOrdenacao”.

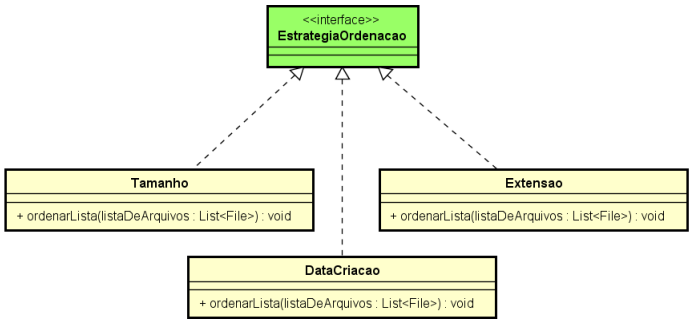

Figura 4: Implementando Ordenações.

Ao implementar a interface ”EstrategiaOrdenacao” o método ordenarLista( ) deve ser sobrescrito. Este método deve conter algum algoritmo de ordenação que, frente a um determinado contexto, será executado para priorizar a transmissão de arquivos de acordo com suas características. A figura 4 ilustra a implementação de três tipos de ordenação: por tamanho, data de criação e extensão de arquivo.

3. Criar classe(s) que implementem a interface “EstrategiaTransmissao”.

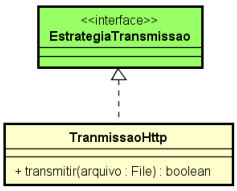

Figura 5: Implementando a transmissão.

Ao implementar a interface ”EstrategiaTransmissao” o método transmitir( ) deve ser sobrescrito. Este método deve conter algum protocolo de transmissão. Podemos ter várias classes que implementam esta estratégia. O método de cada uma destas classes será invocado de acordo com o contexto monitorado. A figura 5 ilustra esta implementação.

4. Estender a classe abstrata “NucleoDeTransmissaoOportunistica”.

   Conforme ilustrado pela figura 6, ao estender a classe ”NucleoDeTransmissaoOportunistica”, os métodos obterAutorizacao( ), obterEstrategiaOrdenacao( ) e obterEstrategiaTransmissao( ) devem ser sobrescritos. O método obterAutorizacao( ) deve conter um algoritmo que represente a lógica para o funcionamento do FOUD. Ao encontrar um lógica que retorne *true* o *framework* será executado. O método obterEstrategiaOrdenacao( ) deve conter lógicas para escolha do algoritmo de ordenação a ser executado em um dato contexto. Cada

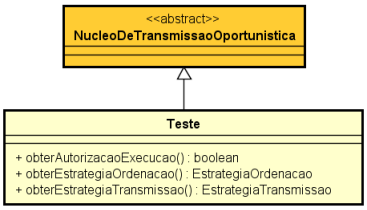

Figura 6: Estendendo a classe NucleoDeTransmissaoOportunistica.

lógica deve retornar uma das classes que implementa a interface ”estrategiaOrdenacao”. De forma semelhante, O método obterTransmissao( ) deve conter lógicas para escolha do algoritmo de transmissão de arquivos a ser utilizado pelo FOUD. Estes algoritmo(s) se encontra(m) em classe(s) que implementa(m) a interface ”estrategiaTransmissao”.

5. Instanciar a classe FOUD no aplicativo A figura 7 ilustra a Criação de um método no aplicativo para que, quando chamado, instancie a classe Foud passando como parâmetro a subclasse de NucleoDeTransmissaoOportunistica.

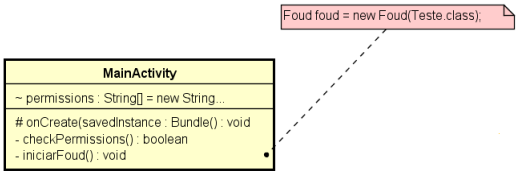

Figura 7: Instanciando o FOUD no aplicativo.

Como forma de testar as funcionalidades do *framework*, foi criado um aplicativo com apenas uma interface (Activity) e um Botão (Button). Arquivos de diferentes tamanhos e extensões foram copiados manualmente para o diretório principal de envio: a pasta ”A enviar”. Ao clicar no botão, o *framework* é executado em *background* e o algoritmo para transmissão dos arquivos para o servidor é iniciado. Seguindo os mesmos passos do início desta seção, vamos apresentar como foi realizada a construção deste aplicativo.

Inicialmente foram criadas duas classes de contexto, Bateria (figura 8) e InterfaceDeRede. Estas classes possuem métodos através dos quais é possivel extrair o nível da bateria do dispositivo e a identificação da interface de rede ativa, respectivamente. O objetivo é criar um pequeno contexto de exemplo, que influencie na tarefa de transmitir arquivos.

```
class Bateria {
    Context context;
    Bateria(Context context) { this.context = context; }

    public int getNivel() {
        IntentFilter intentFilter = new
                IntentFilter(Intent.ACTION_BATTERY_CHANGED);
        Intent batteryStatus = context
                .registerReceiver( receiver: null, intentFilter);
        return batteryStatus
                .getIntExtra(BatteryManager.EXTRA_LEVEL,  defaultValue: -1);
    }
}
```

Figura 8: Criação de classes contexto no Android.

Posteriormente criamos as subclasses de ordenação e transmissão. Como descrito, foram criadas três classes que apresentam diferentes algoritmos de ordenação, sendo as classes Tamanho, DataCriacao e Extensao (figura 4). Como subclasse de transmissão criamos apenas uma, que utiliza o protocolo http.

```
public class Extensao implements EstrategiaOrdenacao {
    private List<String> extensoesPrioritarias;
    public Extensao(List<String> extensoesPrioritarias) {
        this.extensoesPrioritarias = extensoesPrioritarias;
    }

    @Override
    public void ordenarLista(List<File> listaDeArquivos) {
        listaDeArquivos.sort((Comparator) (f1, f2) → {
                String ext1 = getExtension(f1);
                String ext2 = getExtension(f2);
                if(retornaPosicao(extensoesPrioritarias, ext1) >
                        retornaPosicao(extensoesPrioritarias, ext2)){
                    return 1;
                }
                if(retornaPosicao(extensoesPrioritarias, ext1) <
                        retornaPosicao(extensoesPrioritarias, ext2)){
                    return -1;
                }else{
                    return 0;
                }
        });
    }
```

Figura 9: Um dos algoritmos de ordenação utilizados (vista parcial).

As lógicas criadas ao estender a classe abstrata "NucleoDeTransmissaoOportunistica" são apresentadas na figura 10. Posteriormente há uma descrição de cada uma destas lógicas.

```
public class Teste extends NucleoDeTransmissaoOportunistica {
    Context context = Foud.context;
    Bateria bateria = new Bateria(context);
    InterfaceDeRede interfaceDeRede = new InterfaceDeRede(context);

    @Override
    public boolean obterAutorizacaoExecucao() {
        if(bateria.getNivel() > 50)
            return true;
        return false;
    }

    @Override
    protected EstrategiaOrdenacao obterEstrategiaOrdenacao() {
        String[] extensoes = new String[]{"mp4"};
        if(bateria.getNivel() > 50 && bateria.getNivel() <= 60)
            return new Tamanho();
        if(bateria.getNivel() >= 90 && interfaceDeRede.
                getIdentificacao().equals("minhaRede"))
            return new Extensao(Arrays.asList(extensoes));
        return new DataDeCriacao();
    }

    @Override
    public EstrategiaTransmissao obterEstrategiaTransmissao() {
        return new Http();
    }
}
```

Figura 10: Lógicas para transmissão de arquivos no Android.

- Lógica de funcionamento: Criamos uma lógica que faça com que o funcionamento do *framework* fique sujeito a nível de bateria acima de cinquenta por cento.
- Lógica de ordenação: criamos uma lógica garatindo que, em caso de nível de bateria maior que cinquenta e menor ou igual a sessenta, a estratégia de ordenação seja de "tamanho", com disposição de arquivos na fila por ordem ascendente.

  Se o nível da bateria for maior ou igual a noventa, e tivermos uma interface de rede com identificação (SSID) predeterminada, a estratégia de ordenação utilizada será a de "extensão de arquivo", onde arquivos com extensões de vídeo "mp4" estejam dipostos nos primeiros lugares na fila.

  Para os demais casos, a estratégia de ordenação utilizada será a de "data de criação", com disposição de aquivos na fila por ordem ascendente.
- Lógica de transmissão: aqui, estamos retornando o protocolo http para qualquer caso.

Através deste teste foi possível simular uma aplicação real, com *upload* de dados de acordo com as regras estabelecidas. O banco de dados começou a ser alimentado e proporcionará execução de estratégias mais inteligentes.

6. CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este trabalho apresentou a proposta de um *framework* que pode ser utilizado por qualquer entidade corporativa ou governamental com interesse em transmissão oportunística e ciente de contexto de dados móveis, coletados em localidades com acesso inexistente ou intermitente com a Internet. Foi apresentado o modelo genérico deste *framework*, que é extensível e, portanto, adaptável, no que diz respeito à escolha dos tecnologias de transmissão, lógica de envio e contextos a serem utilizados. Além disso, apresentou-se também a implementação deste modelo para o sistema Android, bem como o uso do mesmo no desenvolvimento de um aplicativo de exemplo.

Podemos considerar para trabalhos futuros:

- Implementação de mecanismos de inferência, de modo a permitir a transmissão mais inteligente de arquivos baseando-se nos dados contextuais acumulados pelos aplicativos que utilizam este *framework*. Pacotes de arquivos poderiam ser formados para uma transmissão mais oportunística, utilizando previsões.
- Adaptação do *framework* para possibilitar trabalho com sites offline, como se estivéssemos online, no que se refere ao preenchimento de formulários.
- Permição de modificações sobre os arquivos antes da transmissão. Desta forma poderíamos escolher alguns arquivos, de acordo com suas características (nome ou extensão, por exemplo), compactá-los e criptografá-los antes de transmití-los.

7. REFERÊNCIAS

[1] https://www.dropbox.com. Acesso em: 06 de dez. de 2018.

[2] https://www.google.com/drive. Acesso em: 06 de dez. de 2018.

[3] https://onedrive.live.com. Acesso em: 06 de dez. de 2018.

[4] httpshttps://developer.android.com/topic/libraries/architecture/workmanager. Acesso em: 15 de dez. de 2018.

[5] N. Aminzadeh, Z. Sanaei, and S. H. Ab Hamid. Mobile storage augmentation in mobile cloud computing: Taxonomy, approaches, and open issues. *Simulation Modelling Practice and Theory*, 50:96–108, 2015.

[6] M. Bazire and P. Brézillon. Understanding context before using it. In *International and Interdisciplinary Conference on Modeling and Using Context*, pages 29–40. Springer, 2005.

[7] E. Benkhelifa, T. Welsh, L. Tawalbeh, Y. Jararweh, and A. Basalamah. User profiling for energy optimisation in mobile cloud computing. *Procedia Computer Science*, 52:1159–1165, 2015.

[8] R. E. Kraut, R. S. Fish, R. W. Root, and B. L. Chalfonte. Informal communication in organizations: Form, function, and technology. In *Human reactions to technology: Claremont symposium on applied social psychology*, pages 145–199, 1990.

[9] Y. Li, M. Qian, D. Jin, P. Hui, Z. Wang, and S. Chen. Multiple mobile data offloading through disruption tolerant networks. *IEEE Transactions on Mobile Computing*, 13(7):1579–1596, 2014.

[10] B. Zhou, A. V. Dastjerdi, R. N. Calheiros, S. N. Srirama, and R. Buyya. A context sensitive offloading scheme for mobile cloud computing service. In *Cloud Computing (CLOUD), 2015 IEEE 8th International Conference on*, pages 869–876. IEEE, 2015.