Uma Proposta de *Middleware* para Envio e Recebimento de Mensagens de Objeto (*Intents*) através de *Sockets*: Estudo de Caso do PROGMA

Michelle G. C. Silva[1], Leonardo O. Moreira[2], Emanuel F. Coutinho[2]

[1]Programa de Pós-Graduação em Engenharia de Teleinformática (PPGETI)
Universidade Federal do Ceará (UFC) – Fortaleza, CE – Brasil

[2]Instituto Universidade Virtual (UFC Virtual)
Universidade Federal do Ceará (UFC) – Fortaleza, CE – Brasil

michellecacais@gmail.com, {leoomoreira,emanuel}@virtual.ufc.br

Abstract. *In this work, we present a proposal of a middleware to connect two or more devices using sockets and an endpoint of a communication process across a network. The greatest contribution of using this technology is the possibility of transporting messaging object to the receiving device. Furthermore, we introduce a case study based on the features of this middleware: Program for Management and Sharing Files (PROGMA). This consists of an appliance for smartphones and tablets which operating system is Android. It aims to send and open digital files through the network to other Android devices. This program was designed for teachers to control what students are seeing in their decives. Throughout this document, it will be shown how this middleware works, major components that comprise it, the architectural model adopted, operation of PROGMA and its performance test. In addition, we make a brief overview of middleware usage as part of other programs.*

Resumo. *Neste trabalho, apresentamos uma proposta de middleware para conectar dois ou mais dispositivos usando sockets de rede e um endpoint de um processo de comunicação através de uma rede. A maior contribuição da utilização desta tecnologia é a possibilidade de transportar objetos de mensagens para o dispositivo receptor. Além disso, apresentamos um estudo de caso com base nas características deste middleware: Program for Management and Sharing Files (PROGMA). Este é constituído por uma aplicação para smartphones e tablets cujo sistema operacional é o Android. Destina-se a enviar e abrir arquivos digitais através da rede para outros dispositivos Android. Este programa foi concebido para que professores possam controlar o que os alunos estão vendo em seus dispositivos. Ao longo deste documento, será mostrado como este middleware funciona, os principais componentes que o compõem, o modelo arquitetônico adotado, a operação do PROGMA e seu teste de desempenho. Além disso, fazemos uma breve visão geral do uso do middleware como parte de outros programas.*

1. Introdução e Motivação

Um sistema composto por hardware e software localizados em lugares diferentes, que se comunicam através de mensagens enviadas através de uma rede é considerado um

sistema distribuído. Segundo a definição de Tanenbaum (2007), "Sistema Distribuído é uma coleção de computadores independentes que se apresenta ao usuário como um sistema único e consistente" [Tanenbaum e van Steen 2007].

Para organizar a interação entre cada máquina, a camada de comunicação não deve conter informações que não sejam interpretadas por esses. Para resolver esse e outros problemas de integração, um *middleware* pode ser utilizado [Henricksen e Robinson 2006]. Esta tecnologia é uma camada de software que fornece uma abstração de programação. Além de resolver problemas com a heterogeneidade entre as máquinas, este é um modelo operacional uniforme.

Um dos principais desafios para o desenvolvimento de um *middleware* [Pérez e Gutiérrez 2014] é a integração entre máquinas conectadas à Internet. Há uma necessidade de construir algo que facilite a comunicação em rede e abstraia a heterogeneidade dos sistemas. Um bom exemplo de *middleware* que integra sistemas operacionais heterogêneos é o *Common Object Request Broker Architecture* (CORBA). Este fornece aplicativos com uma variedade de serviços, tais como as atribuições de nomes de membros de software, segurança de dados, armazenamento persistente, notificações de eventos, entre outros.

Outro exemplo é o *Java Remote Method Invocation* (RMI), que estende o modelo de objetos Java. Esta tecnologia permite a invocação remota de objetos usando a mesma sintaxe de objetos locais. A programação de aplicações usando Java RMI é relativamente simples, pois utiliza apenas uma linguagem de programação.

A proposta deste *middleware* apresentado neste artigo difere de outros projetos, pois tem como objetivo integrar os dispositivos móveis, permitindo o envio de intenções (*intents*) através de uma rede. Esta proposta foi baseada em conceitos de um sistema colaborativo, que tem como objetivo ajudar a unir as pessoas que estão trabalhando em uma tarefa similar. É basicamente um programa multi-usuários, o que significa que é necessário mais de uma pessoa para ser usado.

O estudo de caso, utilizando neste presente trabalho, consiste em um programa que usa este *middleware* e seu modelo de comunicação: Programa para Compartilhamento e Gerenciamento de Arquivos (PROGMA), uma aplicação para *smartphones* e *tablets* com sistema operacional Android. Esse foi feito para os professores utilizarem em sala de aula para o envio e gerenciamento de arquivos digitais em máquinas dos alunos. Usando este programa, os profissionais da educação poderiam controlar o que aparece nos dispositivos do aluno, proporcionando ainda mais autonomia para eles.

Este trabalho espera mostrar uma nova forma de fazer comunicação entre dispositivos Android, especialmente falando sobre o PROGMA e suas contribuições para a educação, uma vez que o uso de recursos tecnológicos nas escolas tornou-se comum e muito produtivo para aqueles que utilizam esta prática. Há uma série de implementações que podem utilizar este modelo para o envio de mensagens de objeto através de uma rede. Ao longo do trabalho será discutida também a possibilidade de utilização deste *middleware* para trabalhos futuros. Não obstante, serão sugeridas aplicações que poderiam se beneficiar dessa forma de comunicação multi usuários para troca de *intents*.

Buscou-se nesse trabalho idealizar, construir e demonstrar um *middleware* para integrar dois dispositivos Android pela rede. Através da ferramenta proposta, é possível

enviar e receber mensagens de objeto (*intents*) para executarem métodos no aparelho receptor, como abrir a câmera, enviar uma foto ou arquivos digitais, entre outros. O objetivo geral é desenvolver um *middleware* que sirva como canal de comunicação de transporte de *intents* entre aparelhos Android conectados à rede. Para alcançar o objetivo geral, alguns objetivos específicos foram elencados: (a) idealizar uma arquitetura para comunicação entre aparelhos Android para envio e recebimento de *intents* entre máquinas que se comportem como cliente e servidor; (b) criar uma aplicação móvel que utilize essa ferramenta que sirva como caso de estudo; (c) levantar dados sobre o comportamento da aplicação com testes unitários e de escalabilidade; (d) identificar aspectos positivos, negativos e pontos a serem melhorados, garantindo a evolução do sistema.

O artigo está organizado da seguinte forma: a Seção 2 destaca a fundamentação teórica, base para o entendimento de como funciona a tecnologia proposta nesse trabalho. Já a seção 3 demonstra e discute os trabalhos relacionados ao trabalho proposto, destacando as principais características de cada trabalho. Já a Seção 4 apresenta o *middleware* idealizado para transportar mensagens de objeto entre um dispositivo e outro. A Seção 5 introduz o estudo de caso, o PROGMA, mostrando como a tecnologia proposta foi usada e explicitando os resultados alcançados com este por meio de testes de desempenho. Por fim, na Seção 6 são feitas as considerações finais e são destacados os trabalhos futuros.

2. Fundamentação Teórica

Esta seção apresenta a fundamentação teórica utilizada neste trabalho e essencial para compreender como funciona o *middleware* proposto. Conceitos fundamentais sobre redes, camadas de protocolos e modelos de serviços, protocolos de transporte, *sockets* de redes, programação com *sockets* utilizando *Transmission Control Protocol* (TCP) e definições de middleware são apresentados, assim como os modelos existentes na literatura.

2.1. Camadas de Protocolo e Modelos de Serviço

O modelo *Open Systems Interconnection* (OSI) é um modelo padrão para protocolos de comunicação e para garantir a comunicação fim-a-fim [Kurose e Ross 2012]. O modelo OSI divide as redes de computadores em sete camadas criando uma espécie de abstração e definindo o papel de cada camada [Coulouris et al. 2011].

A camada de aplicação é o local onde as aplicações de rede e os protocolos de aplicação de camadas ficam. É nessa também em que os nomes dos domínios, *Domain Name System* (DNS) são transformados em endereços de internet de 32 *bits*. A camada de apresentação atua como intermediária no processo. Essa é responsável pela formatação dos dados e da apresentação destes além de cuidar para que duas redes diferentes se comuniquem. Na camada sessão, é fornecida a delimitação e sincronização da troca de dados, incluindo a intenção de fazer uma inspeção e um esquema de recuperação. O envio dos dados é feito pela camada de transporte. Esta trata do protocolo que será utilizado para executar essa tarefa. Existem dois tipos de protocolos: TCP e UDP, que serão tratados adiante. A camada de rede é responsável por mover os pacotes de dados, também conhecidos como datagramas. Esse protocolo é responsável por providenciar o serviço para entrega dos dados. A camada de enlace destina uma rota ao datagrama entre outras séries de rotas do remetente ao destinatário. Esses dados são tratados pela camada física, que move os *bits* individualmente.

2.2. Protocolos de Transporte

A camada de transporte é responsável por envio dos pacotes de dados (datagramas) e por determinar o protocolo utilizado: *User Datagram Protocol* (UDP) ou *Transmission Control Protocol* (TCP).

O modelo TCP inclui uma conexão orientada ao serviço e uma transferência confiável de dados. Para garantir a confiabilidade, esse protocolo tem as informações do cliente e do servidor antes das informações começarem a ser enviadas. Dessa forma, esse abre a conexão e garante a entrega dos dados. Além disso, TCP provê entrega ordenada com detecção e recuperação de erros. Assim, além de garantir a entrega das informações enviadas, também estipula a integridade e a ordenação dessas. A desvantagem desse protocolo é a lentidão em comparação ao UDP, por conter muitas informações em seu cabeçalho.

Já o UDP não é orientado a conexão. Esse protocolo é muito simples, já que não fornece controle de erros. A entrega de dados não é garantida e nem a ordenação desses, assim como controle de fluxo e congestionamento. O grande benefício dessa tecnologia é a velocidade, por não garantir a chegada dos dados. É recomendado que TCP seja utilizado em aplicações que necessitem de entrega confiável, como envio de e-mails. Por outro lado, UDP é recomendado para projetos que não necessitem de total entrega de dados, e que algumas perdas não venham a ser reparadas ou prejudiciais ao total funcionamento desse, como transmissão de vídeos.

2.3. *Sockets* de rede

Uma aplicação típica de rede de internet consiste em um par de programas: um cliente e um servidor residentes em diferentes sistemas. Esse tipo de arquitetura é definida como uma estrutura de aplicação que distribui as tarefas e cargos de trabalho de fornecedores de serviço (servidores) e chamadas de métodos (clientes). Quando esses dois programas são executados, um processo cliente e um servidor são criados e eles se comunicam através de *sockets*. O *socket* é o ponto final de um fluxo de comunicação através da rede.

Em uma aplicação que utiliza *socket*, um datagrama é transmitido entre processos quando um programa efetua um comando para envio (*send*) e o outro para recebimento (*receive*). Para tanto, é necessário que seja aberta uma porta e que os programas cliente e servidor estejam conectados através dessa. O papel do servidor envolve a criação de um *socket* de "escuta" vinculado à porta de serviço para esperar as requisições de conexão. Já o cliente cria um fluxo vinculado a qualquer porta, e depois um pedido para se conectar (*connect*) em determinada porta.

2.4. Definições de *Middleware*

Desde o advento das redes de computadores locais tem-se desenvolvido sistemas distribuídos utilizando *middleware*. O *middleware* consiste em uma camada de software para facilitar o desenvolvimento e execução de operações em máquinas remotas que se comunicam para compor um sistema como se fosse homogêneo. Uma das principais funcionalidades do *middleware* é dar suporte às diversas características desejáveis para um sistema distribuído, como interoperabilidade, integração, portabilidade, escalabilidade e ajudar na integração entre as diferentes formas de interação entre os sistemas operacionais.

Segundo Coulouris (2011), "(. . .) a tarefa do *middleware* é fornecer uma abstração de programação de nível mais alto para desenvolvimento de sistemas distribuídos". O que o autor afirmou nesse excerto foi que o propósito dessa tecnologia é abstração da complexidade para integração das partes remotas do sistema. A partir dessa noção, é possível dizer que o principal intuito dessa camada de software é "mascarar" a complexidade de aplicações distribuídas e facilitar a integração de sistemas operacionais distintos.

Os serviços oferecidos pelo *middleware* possuem propósito geral, situados entre plataforma e aplicações sendo caracterizado pela Interface de Programação da Aplicação (API - *Application Programmer Interface*), isto é, um conjunto de rotinas e padrões descritos por um software, e pelos protocolos que suporta. O *middleware* oferece facilidades para apoiar diferentes tipos de colaboração entre os sistemas [Henricksen e Robinson 2006]. Também oferece flexibilidade para apoiar sistemas colaborativos entre diferentes plataformas de hardware e software.

Essa tecnologia pode oferecer serviços como gerenciamento de apresentação e informação, comunicação, controle, gerenciamento de sistemas e entrega, entre outros. Existem diversos modelos de *middleware* que são baseados em diferentes características, como tipo de comunicação, linguagem para construção da aplicação, forma de disponibilização, ambiente de execução, entre outros. Para este trabalho, a classificação de Talarian (2000) foi escolhida por ser mais específica em relação aos parâmetros escolhidos para categorização [Talarian 2000].

2.5. Sistema Operacional Android

O Android [Android 2016a] é um sistema operacional para tecnologia móvel baseado em uma modificação do sistema Linux. Foi originalmente idealizado por um *startup*, empresa com nível operacional limitado, de mesmo nome. Em 2005, a companhia Google comprou o sistema e assumiu o desenvolvimento a partir desse momento. Sua principal característica do Android é ter o código aberto. Isso quer dizer que qualquer pessoa que quiser usar o sistema pode fazer o *download* do código completo.

Para entender como o Android funciona, é importante saber as camadas que o compõem. As aplicações e os *frameworks* rodam sobre uma máquina virtual chamada "Dalvik". A estrutura do Android é basicamente dividido em cinco sessões e em quatro camadas principais: Linux Kernel, bibliotecas (*libraries*), tempo de execução do Android (*Android runtime*), *framework* de aplicações (*application framework*) e aplicações (*applications*).

O Linux Kernel é o núcleo que o Android é baseado. Nessa camada estão todos os *drivers* de dispositivo de baixo nível [Sarma et al. 2012]. As bibliotecas contém todo o código que fornece os principais recursos do sistema operacional. O tempo de execução está na mesma camada das bibliotecas e fornece um conjunto de bibliotecas de núcleo que permite aos desenvolvedores escrever aplicativos em linguagem Java. O *framework* das aplicações expõe as várias capacidades do sistema operacional. Na camada de aplicações estão os aplicativos oriundos do Android, como o navegador, o telefone, os contatos, e outros que podem ser baixados pela loja do sistema. A plataforma Android oferece suporte aos desenvolvedores, disponibilizando o *kit* de desenvolvimento de software, ou *Software Development Kit* (SDK), um pacote com as ferramentas e bibliotecas necessárias para criação, teste e *debug* de aplicativos.

2.5.1. Intenções (*Intents*)

Quando se discute sobre aplicações Android, esta sempre se inicia por uma *Activity*, que é uma janela que contém a interface do usuário. Para navegar entre uma tela da aplicação Android (*Activity*) e outra, é necessário o uso de *intents* [Jing et al. 2016]. Este é uma descrição abstrata de uma operação a ser realizada, funcionando como uma "cola" para ligar as atividades da aplicação. É basicamente uma estrutura de dados passiva que suporta a descrição de uma ação a ser executada. *Intents* permitem interação com outros componentes da aplicação, por exemplo, uma atividade pode iniciar um programa externo para tirar uma foto ou executar outras tarefas.

Intent é um recurso muito importante do Android, pois através desse é possível fazer as aplicações se comunicarem, disponibilizando ações que podem ser reutilizadas, sem a necessidade de importar dependências ou bibliotecas para o projeto. São criadas geralmente a partir de ações do usuário e representam a intenção de se realizar algo. Podem ser definidas como mensagens enviadas por um componente da aplicação informando a intenção de inicializar alguma ação. A Figura 1 apresenta um exemplo de um *intent* conectando duas *activities*.

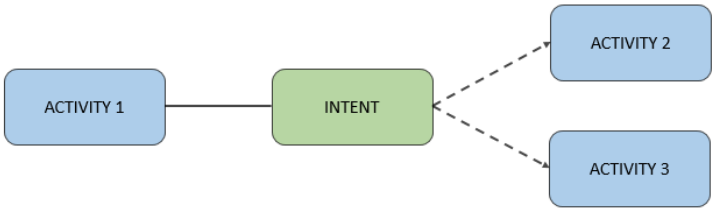

Figura 1. Ilustração de um *intent* conectando duas *activities*

Existem alguns tipos de *intents* para executar diferentes tarefas. Usando a ideia de transporte pela rede, é possível escolher e enviar essas mensagens de objeto para o aparelho receptor. Em uma aplicação pode ser definido o componente de destino diretamente na intenção (*explicit intent*) ou requerer que o sistema Android avalie componentes registrados com base nos dados desse (*implicit intent*). *Explicit intents* definem especificamente o componente que deve ser chamado usando a classe Java como identificador. Já *implicit intents* especificam as ações que devem ser realizadas e os dados que fornecem o conteúdo da ação. Em resumo, *intent* é o conjunto de informações necessárias para ativar um componente de aplicação. Conhecer o comportamento deste é muito importante para desenvolvimento de aplicações Android, pois permite o reaproveitamento de funcionalidades que já estão prontas na plataforma.

3. Trabalhos Relacionados

A ideia de integrar dois aparelhos pela rede não é nova e já existem vários trabalhos que estudaram formas de abrir um canal de comunicação para envio e recebimento de dados. Dentre os que mais se adequam à discussão proposta podem ser citados os seguintes:

- *Distributed Intent*: *Android Framework for Networked Devices Operation*, em português, "*Intent* Distribuído: *Framework* para Operações em Dispositivos Android conectados à Rede" [Nagahara et al. 2013]: nesse trabalho uma estrutura

distribuída que estende *intents* para melhorar a colaboração entre máquinas com sistema Android é proposta. No quadro proposto, dispositivos colaboram com sistemas embarcados enviando intenções serializadas.

- *Analyzing Inter-Aplication Communication in Android*, em português, "Analisando comunicação inter-aplicações em Android" [Chin et al. 2011]: nesse trabalho as interações entre aplicações Android são examinadas e são identificados riscos de segurança em componentes da aplicação. É fornecida uma ferramenta chamada ComDroid, que pode ser usada por desenvolvedores para analisar a vulnerabilidade em seus próprios aplicativos antes de irem para o Google Play, ferramenta de compras de ferramentas para Android.
- *User Intent to Support Proactivity in a Pervasive System*, em português, "Intenção do usuário para dar suporte à proatividade em um sistema pervasivo" [Shaaban et al. 2009]: o objetivo deste trabalho é apresentar alguns aspectos de como *intents* do usuário podem ser manuseados, com foco na arquitetura que dá suporte às ferramentas de plataformas pervasivas.

Os trabalhos relacionados apresentam diferentes formas de estudar *intents* e manuseá-los. O que mais se aproxima da proposta de *middleware* deste trabalho é o framework para distribuir *intents* pela rede. A principal diferença é que o foco do artigo é a comunicação entre aparelhos Android e sistemas embarcados, enquanto que no caso do *middleware* proposto seria apenas entre os dispositivos com esse sistema operacional.

4. Proposta de *Middleware* para Transportar *Intents*

Esta seção discorre sobre a possibilidade de transportar mensagens de objetos (*intents*) entre dois ou mais dispositivos Android. A ideia para compor o *middleware* proposto é utilizar uma conexão com *sockets* usando TCP. Desse modo, é possível enviar e receber intenções para passar de uma ação a outra nos componentes do aplicativo.

A navegação entre uma *Activity* e outra é feita através de *intents*. Os *intents* são usados para iniciar uma *Activity*, para disparar sub-atividades, isto é, atividades iniciadas por outras atividades, e iniciar serviços. A ideia principal do *middleware* proposto é poder enviar essas mensagens para outros dispositivos e assim possibilitar que outros programas sejam abertos no aparelho que receba a intenção. Desse modo, uma ação poderia ser enviada para outro aparelho executar sua câmera eregistrar umas fotografia.

O papel do *middleware* é garantir que essas mensagens sejam passadas de um aparelho para outro. O desenvolvedor teria liberdade para escolher quais tipos de *intents* serão executados na máquina que irá recebê-los. Ao abrir a conexão para envio das mensagens, é possível enviar as intenções e abrir a agenda, buscar as coordenadas de localização por *Global Positioning System* (GPS), abrir uma página *web*, entre outras ações, no aparelho receptor.

Utilizando esse recurso, diversos tipos de *intents* podem ser enviados para dispositivos conectados à rede. Por exemplo, iniciar outros aplicativos como a localização GPS, reproduzir automaticamente uma música ou abrir um arquivo na máquina que está recebendo essas intenções iniciadas por outra máquina. Além disso, os aplicativos instalados no dispositivo também podem ser acessados por *intents*, como acessar a câmera e requisitar o resultado para ser processado por um método criado pelo programador.

A vantagem dessa forma de comunicação entre aparelhos Android é a segurança, pois o protocolo utilizado é o TCP, ou seja, existe a garantia de que os pacotes de dados serão entregues ao sistema final. Além disso, a programação deste é relativamente simples e fácil de ser customizada, pois após programar os *sockets* é necessário apenas escolher o tipo de *intent* que deseja enviar pela rede. Como desvantagem, pode ser citado que esse *middleware* não funcionará com internet de terceira geração da telefonia móvel (3G). Os dispositivos devem estar conectados à mesma rede, caso contrário, não haverá possibilidade de estabelecer conexão.

4.1. Visão Geral do Funcionamento da Proposta

Para realizar o transporte de *intents* pela internet, é necessária a utilização de *sockets* de rede. O protocolo utilizado para este *middleware* é o TCP, pois implementa um canal confiável, isto é, garante a entrega e a ordenação dos dados enviados. Além disso, o TCP possui controle de fluxo, onde à medida em que o receptor recebe os dados, ele vai enviando mensagens *Acknowledgement* (ACK) confirmando que este foi entregue.

Como foi visto na revisão da literatura, é necessário um lado cliente e um servidor ao criar uma conexão *socket*. A proposta é fazer com que dois dispositivos se conectem e se comportem em um momento como clientes, e em outro momento como servidores, dependendo de sua configuração. Deixando claro que o lado servidor da aplicação irá receber os *intents*, e o lado cliente irá enviá-los.

A construção do *middleware* proposto é simples de ser implementada. Após criar o projeto Android, deve-se codificar o *socket* em linguagem Java, que segue o modelo clássico apresentado na revisão da literatura, mais precisamente na seção 2.3. Como o protocolo utilizado é o TCP, para que os aparelhos possam se comunicar e enviar dados um para o outro, é necessário que os lados cliente e servidor se apresentem e estabeleçam uma conexão.

A ideia principal dessa aplicação é abrir um canal de comunicação entre os dois aparelhos, sendo um deles o cliente e o outro o servidor. O servidor abre uma porta na rede e "escuta" os programas do lado cliente ativo que podem fazer uma requisição de conexão. Quando contactado pelo cliente, o servidor cria um novo *socket* para se comunicar com esse. Ao final desse processo, é enviado o *intent* contendo uma ação a ser executada na máquina do servidor. Logo em seguida a conexão socket pode ser encerrada em ambos os lados. A Figura 2 mostra visualmente o procedimento descrito com um diagrama de sequência.

Para a comunicação entre dois aparelhos Android, é necessário explicar que o servidor deve abrir a porta e esperar o cliente se conectar. Para ficar mais claro o entendimento dessa proposta de *middleware*, será apresentado na Figura 3 um trecho de código mostrando como enviar uma *intent* para abrir uma página específica no navegador padrão do dispositivo servidor.

Esse trecho de código mostra como passar um endereço para ser aberto no navegador do aparelho receptor da intenção, ou seja, o servidor. Primeiro é realizada a aceitação do cliente *socket* e abre-se um canal para leitura de dados. Em seguida é enviado um *intent* contando as instruções para abrir o portal da Universidade Federal do Ceará (UFC)[1]

[1]Universidade Federal do Ceará (UFC) - http://www.ufc.br

na máquina.

Isso conclui a apresentação geral do funcionamento do *middleware*. Espera-se que tenha ficado clara a explicação sobre a arquitetura e a utilização deste com o modelo de código apresentado.

5. Estudo de Caso: PROGMA

Esta seção apresenta um caso de estudo que utiliza o *middleware* proposto: o Programa para Gerenciamento e Compartilhamento de Arquivos (PROGMA). Trata-se de um aplicativo para dispositivos Android que permite o envio de arquivos digitais através da rede e a opção de abrí-los na máquina receptora. Não obstante, serão discutidos temas fundamentais para o entendimento de como a aplicação funciona, como sua arquitetura, seus componentes, principais trechos de código e os testes e resultados.

O PROGMA, surgiu com o intuito de proporcionar uma opção para controle de documentos utilizados em sala de aula. É importante ressaltar também que não só profissionais da área de ensino poderão se beneficiar do projeto. Este poderá ser usado em diferentes ocasiões, sendo uma alternativa para que imagens, documentos de texto e arquivos de diferentes formatos sejam enviados de um aparelho para outro.

A aplicação permite o compartilhamento de arquivos em formato digital através da rede e a opção para abri-los no aparelho receptor. Dessa forma, é possível que a máquina cliente, no caso o professor, envie imagens, vídeos e textos em formato compacto de documento, em inglês, *Portable Document Format* (PDF) para a máquina servidora, no

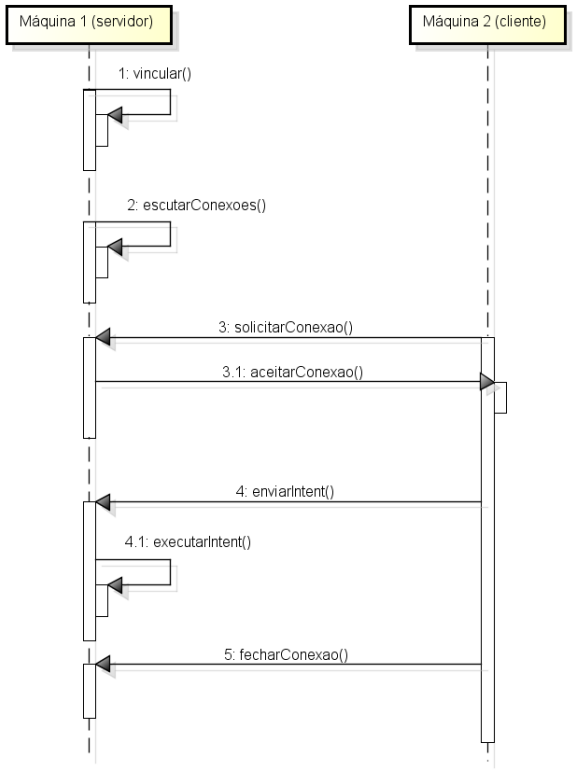

Figura 2. Diagrama de sequência do *middleware*

```
try {

    clientSocket = serverSocket.accept();
    BufferedReader inFromClient = new BufferedReader(new InputStreamReader(clientSocket.getInputStream()));
    String buffer = inFromClient.readLine();
    if(!buffer.equals("ping")){
        if(buffer.equals("conectar")){
            System.out.println(buffer);
            String url = "http://www.ufc.br";
            Intent intent = new Intent(Intent.ACTION_VIEW);
            intent.setData(Uri.parse(url));
            startActivity(intent);
        }
    }
    else    System.out.println(buffer);

} catch (IOException ex) {
    System.out.println("Problem in message reading");
}
```

Figura 3. Trecho de código de envio de *intent* para abrir uma página *web*

caso, os *tablets* ou *smartphones* dos alunos, e os abra por meio de um comando enviado para estes. Essa medida garante mais autonomia do docente, pois este estará controlando o que aparece nos aparelhos dos estudantes e não precisará pedir verbalmente para que esses abram e visualizem os arquivos.

Além da possibilidade de transferir documentos digitais, o sistema permite ainda a criação de grupos, por exemplo, 7º ano, 8º ano, entre outros, para facilitar a localização dos dispositivos na rede que o professor queira conectar e evitar enviar arquivos para aparelhos conectados à rede que não sejam do grupo escolhido. Para compreender melhor uma das funcionalidades do PROGMA, observe a Figura 4.

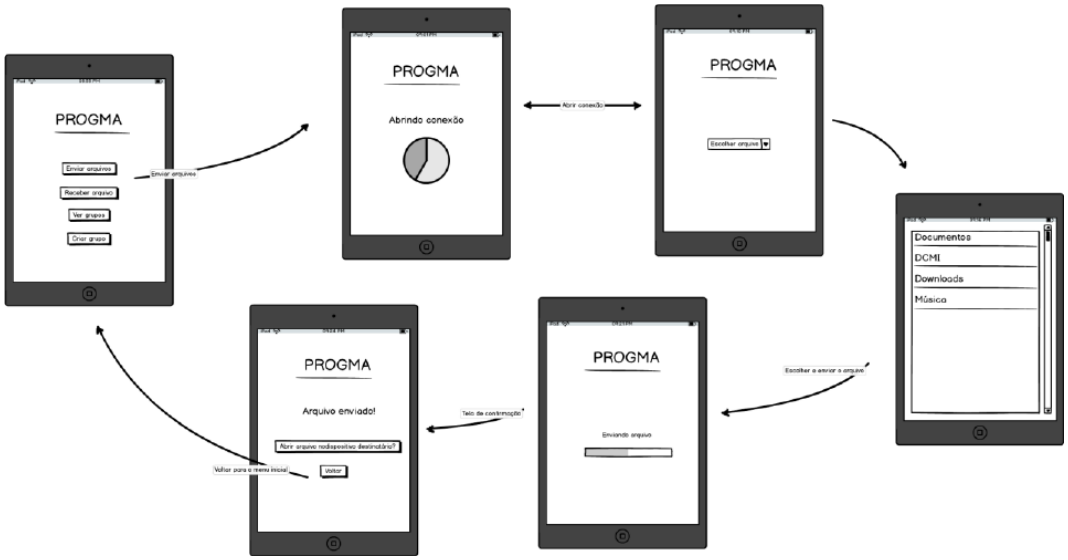

Figura 4. Demonstração de envio de arquivo pelo PROGMA

A Figura 4 mostra um diagrama contendo os passos que o professor deverá fazer para enviar arquivos para as máquinas dos alunos. As opções iniciais do programa são "Enviar arquivo", que abre uma conexão cliente por *socket*, "Receber arquivo", que esta-

belece um servidor *socket* e espera a conexão do cliente para criar o canal de comunicação, "Ver grupos", com a possibilidade de visualizar os grupos criados e enviar arquivos para esses, e "Criar grupo", que permite cadastrar os números dos aparelhos que compõem determinado grupo, no caso do ambiente escolar, as séries.

O programa foi feito em linguagem Java utilizando o SDK [Android 2016a] para Android disponibilizado pela Google, contendo as ferramentas e bibliotecas necessárias para criação de aplicativos. A IDE utilizada foi Android Studio [Android 2016b], também fornecida pela Google. Até o momento da entrega desse trabalho, está em versão beta, mas já apresenta uma rica plataforma para codificação, resolvendo muitos problemas que existem nas atuais opções.

Para a criação de grupos, optou-se por utilizar a base de dados SQLite [SQLite 2016]. O SDK possui um pacote com as classes para gerenciar o próprio banco de dados privado. Com essa biblioteca, é possível fazer as chamadas SQL mais comuns em banco de dados (inserir – *insert*, atualizar – *update*, deletar – *delete* e selecionar – *select*) de uma forma simples e familiar aos programadores. É uma boa opção para persistência de dados e não há necessidade de instalação ou configuração, por esses motivos foi escolhido para criação de grupos.

Como resultado, obteve-se um protótipo funcional para enviar arquivos e abrí-los no dispositivo receptor. A ideia principal era testar se o *middleware* realmente funcionaria, por isso não houve muita preocupação com a aparência do programa. Como trabalhos futuros, poderiam ser citados a melhoria do *design*, tornando-o mais atrativo visualmente, e trabalhar a usabilidade, pois não foi pensada nessa primeira versão do programa.

Figura 5. Tela inicial do aplicativo

A Figura 5 apresenta a atividade principal do programa (*Main Activity*), com as opções de compartilhamento. Se o usuário quiser transferir o arquivo para apenas um dis-

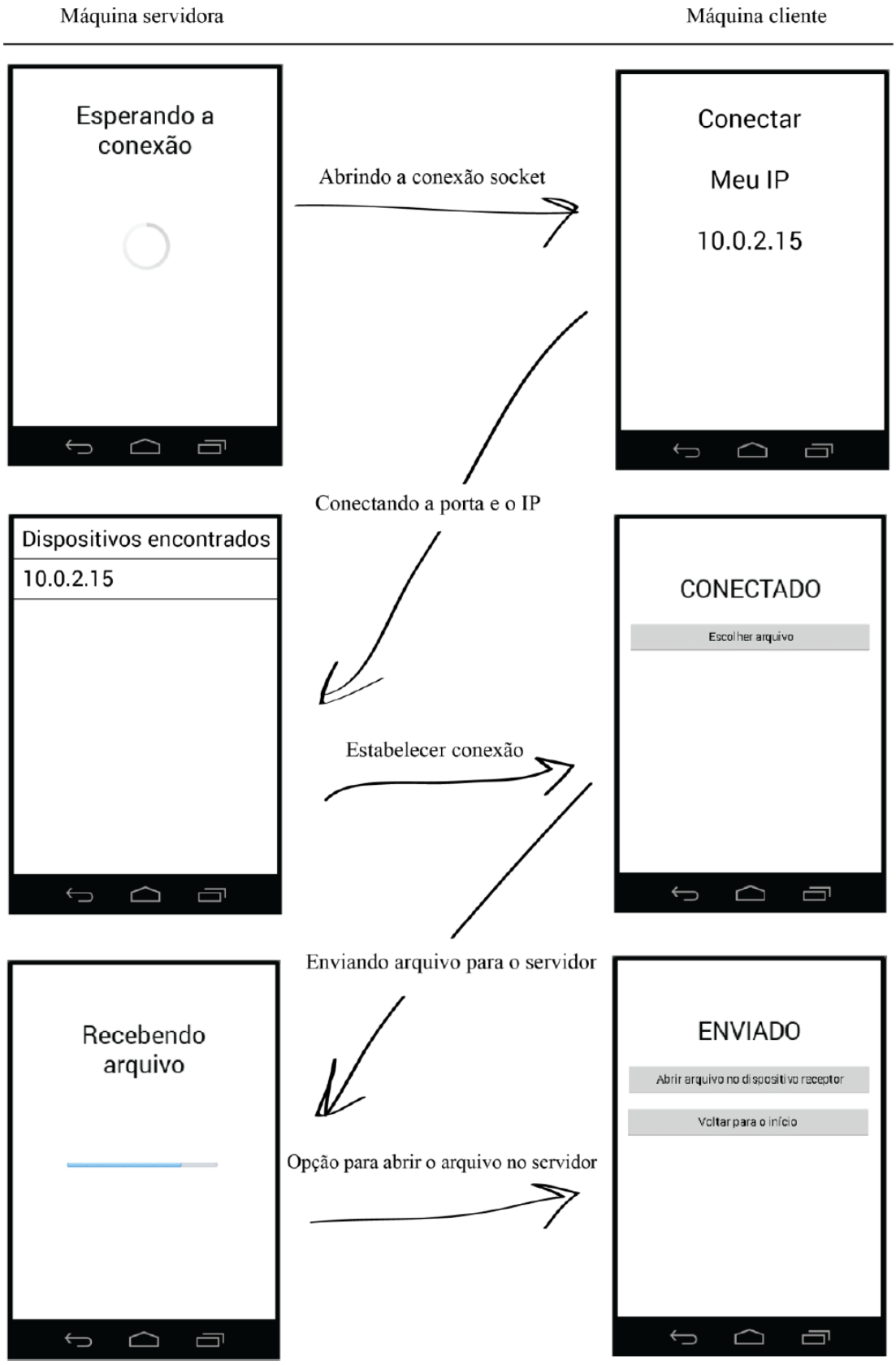

Figura 6. Interação entre cliente e servidor no PROGMA

positivo, ele deve clicar em “Enviar arquivo”, que abrirá a conexão, enviará o documento e possibilitará a abertura deste na máquina receptora. Se existir a necessidade de enviar para mais de um usuário, é recomendável que crie um grupo, dessa forma, os arquivos são transferidos para os integrantes cadastrados.

A Figura 6 exibe as telas de envio de arquivo para apenas um usuário. A interação é bem simples: a máquina servidora abre uma porta na rede e “espera” a conexão do cliente, que por sua vez envia o número IP que é listado no dispositivo servidor. Após estabelecer um canal de comunicação, é enviado o arquivo. Há a possibilidade de abrir o documento enviado se o remetente desejar. Para saber se deve ou não fechar a conexão, foi criada uma variável lógica para saber se é necessário abrir o arquivo no dispositivo receptor e fechar a conexão em seguida, ou fechar a conexão e voltar para a tela inicial. Não é possível enviar o intente para iniciar o documento depois que a conexão é fechada.

Para saber qual programa é o mais adequado para abrir o arquivo na máquina receptora, o usuário cliente deve escolher antes qual o tipo de documento está tentando enviar. Após enviar, existe uma estrutura de decisão para decidir o programa padrão da máquina servidora para executar o arquivo recebido.

Nessa primeira versão foram testados imagens, vídeos e PDF. No caso do grupo, a interação é um pouco mais complicada, pois os números IPs são cadastrados e a máquina que irá enviar o arquivo deve se comportar primeiramente como servidora para estabelecer a conexão. Depois, é enviada para outra *Activity*, em que desempenha papel de cliente e envia o arquivo para a outra máquina que, por sua vez, deverá ter sido enviada também para outra *Activity* e se comportará como servidora. O algoritmo para abertura de conexão pode ser observado na Figura 7.

```
Início
   -> Fazer a consulta no banco de dados
   -> Para i até tamanho do bd faca {
        Consultar endereço IP;
        Abrir conexão socket;
        Enviar arquivo;
        Se (optar por abrir na máquina receptora){
            Enviar intent para abrir arquivo;
            Fechar conexão;
        }Senão{
            Fechar conexão;
        }
   }
Fim
```

Figura 7. Algoritmo de envio de arquivo para grupo pelo PROGMA

Esse tipo de algoritmo não é muito eficaz para enviar arquivos para um grupo muito grande. Dependendo do número de máquinas cadastradas, o sistema acaba sendo bastante custoso. Este é outro ponto a ser melhorado no programa: propor uma otimização para agilizar esse processo. Por fim, para resumir a discussão acerca do sistema, pode-se dizer que essa versão inicial cumpriu o propósito ao testar o *middleware* proposto e iniciar estudos sobre o transporte de *intents* por *sockets* de rede. Ainda falta realizar melhorias para implementar uma futura versão do programa, como mudar o *design*, trabalhar a usabilidade desse protótipo e otimizar o algoritmo de envio de arquivos. Dessa forma, esses aperfeiçoamentos ficam para trabalhos futuros.

5.1. Avaliação

Para obter os resultados do desempenho do programa e verificar se atingiu as especificações, foram feitos testes de sistema, pelo qual são localizadas falhas por meio da utilização do mesmo, como se fosse um usuário final. Dessa forma, os testes são executados com as mesmas condições e mesmos dados de entrada que um usuário utilizaria no cotidiano de utilização do software.

Foram escolhidos três arquivos: uma imagem, um vídeo e um PDF, com tamanhos de 614 KB, 23,4 MB e 177 KB respectivamente. Os equipamentos utilizados foram três *tablets* da marca Samsung Galaxy Tab e dois *smartphones* Moto G da Motorola. Todos possuem sistema operacional Android 4.4. Antes de realizar os testes, levantou-se a hipótese de que o programa se comportaria de maneira funcional, mas com desempenho abaixo do desejável para envio de documentos para grupos, devido o algoritmo não ter sido melhorado. Ainda não foram feitos testes de usabilidade, pois como discutido anteriormente, essa característica não foi trabalhada nesse protótipo inicial.

O envio de arquivos de um único aparelho para outro (1 para 1) foi satisfatório, não excedendo 1 minuto na taxa de transferência de pacotes para a imagem e o PDF. Já o vídeo demorou um pouco mais, mas atingiu um tempo razoável de transmissão. Já com grupos, o desempenho ficou abaixo do desejável, por abrir a conexão e enviar os arquivos de um por um. O tempo de resposta foi superior a 1 minuto por dispositivo e mais de 2 minutos para o vídeo para cada uma das máquinas cadastradas no grupo.

Apesar de esse protótipo inicial ter um desempenho custoso, destaca-se que o *middleware* funciona e pode ser melhorado. Em futuras versões do aplicativo, espera-se que essas falhas sejam corrigidas e que o programa possa ser disponibilizado para download na Play Store.

6. Considerações Finais

6.1. Conclusão

A comunicação entre dois computadores não é igual à interação entre as pessoas. Existe uma dificuldade grande em integrar duas máquinas conectadas a uma rede, inclusive alguns problemas a serem resolvidos, como a heterogeneidade dos sistemas. A partir dessa observação, pode-se dizer que o *middleware* é um paradigma para resolver o problema da complexidade da integração de dois sistemas diferentes. Assim como facilitar o desenvolvimento de sistemas distribuídos.

O *middleware* proposto visa tratar de alguns dos problemas de integração entre máquinas, possibilitando compartilhamento de dados entre um dispositivo e outro e o gerenciamento destes na máquina do destinatário. O estudo de caso mostrou que é possível criar programas com esse *middleware* e utilizá-lo em outras aplicações. A utilização deste se mostrou funcional e cumpriu o prometido, permitindo o envio de arquivos e a abertura deste em dispositivos conectados a uma mesma rede.

Pode-se dizer que ainda existe trabalho a ser feito para melhorar o protótipo, mas com esse caso de estudo já fica provado que o transporte de mensagens de objetos (*intents*) funciona e pode ser usado em outras aplicações.

6.2. Trabalhos Futuros

A principal característica desse modelo proposto é a necessidade de haver dois ou mais dispositivos envolvidos, caracterizando um sistema colaborativo. Ao se pensar em trabalhos que se utilizem do *middleware*, é importante ter em mente que o aplicativo desenvolvido com este é um software que trabalha com colaboração. Tendo isso em mente, uma série de sugestões de aplicações pode ser listada:

- Recurso de pintura colaborativa: aplicação que permite dois ou mais usuários construírem figuras ou ilustrações simultaneamente utilizando tela compartilhada;
- Construção de *storyboard* interativo: sistema que possibilita o compartilhamento de *frames*, ilustrações, balões, entre outros recursos gráficos, para construção de *storyboards*;
- Jogo de pontuação por aproximação: modelo de interação pela qual os usuários só pontuariam se estivessem conectados à mesma rede de outros jogadores. A pontuação seria feita através de troca de métodos pelos *intents*.

Referências

Android (2016a). Android developers. Disponível em `https://developer.android.com/index.html`. Acessado em: 10 de outubro de 2016.

Android (2016b). Android studio. Disponível em `https://developer.android.com/sdk/installing/studio.html`. Acessado em: 10 de outubro de 2016.

Chin, E., Felt, A. P., Greenwood, K., e Wagner, D. (2011). Analyzing inter-application communication in android. In *Proceedings of the 9th International Conference on Mobile Systems, Applications, and Services*, MobiSys '11, pages 239–252, New York, NY, USA. ACM.

Coulouris, G., Dollimore, J., Kindberg, T., e Blair, G. (2011). *Distributed Systems: Concepts and Design*. Addison-Wesley Publishing Company, USA, 5th edition.

Henricksen, K. e Robinson, R. (2006). A survey of middleware for sensor networks: State-of-the-art and future directions. In *Proceedings of the International Workshop on Middleware for Sensor Networks*, MidSens '06, pages 60–65, New York, NY, USA. ACM.

Jing, Y., Ahn, G.-J., Doupé, A., e Yi, J. H. (2016). Checking intent-based communication in android with intent space analysis. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '16, pages 735–746, New York, NY, USA. ACM.

Kurose, J. F. e Ross, K. W. (2012). *Computer Networking: A Top-Down Approach (6th Edition)*. Pearson, 6th edition.

Nagahara, Y., Oyama, H., Azumi, T., e Nishio, N. (2013). Distributed intent: Android framework for networked devices operation. In *2013 IEEE 16th International Conference on Computational Science and Engineering*, pages 651–658.

Pérez, H. e Gutiérrez, J. J. (2014). A survey on standards for real-time distribution middleware. *ACM Comput. Surv.*, 46(4):49:1–49:39.

Sarma, B. P., Li, N., Gates, C., Potharaju, R., Nita-Rotaru, C., e Molloy, I. (2012). Android permissions: A perspective combining risks and benefits. In *Proceedings of the 17th ACM Symposium on Access Control Models and Technologies*, SACMAT '12, pages 13–22, New York, NY, USA. ACM.

Shaaban, Y. A., McBurney, S., Taylor, N., Williams, M. H., Kalatzis, N., e Roussaki, I. (2009). User intent to support pro-activity in a pervasive system. In *PERSIST Workshop on Intelligent Pervasive Environments*, pages 3–8.

SQLite (2016). Sqlite. Disponível em `https://www.sqlite.org/`. Acessado em: 10 de outubro de 2016.

Talarian (2000). Everything you need to know about middleware. Disponível em `http://cdn.ttgtmedia.com/searchWebServices/downloads/Talarian.pdf`. Acessado em: 10 de outubro de 2016.

Tanenbaum, A. e van Steen, M. (2007). *Distributed Systems: Principles and Paradigms*. Pearson Prentice Hall.