CONTROLE E AQUISIÇÃO DE DADOS EXPERIMENTAIS COM TECNOLOGIA *BLUETOOTH* EM DISPOSITIVOS MÓVEIS

[1]Ronaldo dos Santos **Pereira**

[1]Luis Carlos Origa de **Oliveira**

[1] Universidade Estadual Paulista, 15385-000, Ilha Solteira-SP, Brasil

RESUMO

Este artigo apresenta o desenvolvimento do aplicativo AppAquisBluet, utilizando técnicas e métodos disponibilizados na plataforma Android. Sendo, entre varias disponibilizadas no mercado a mais disseminada no mundo, por ser uma plataforma moderna, flexível e *open source*, e possuir ótima relação custo/benefício, ideal para a implementação do projeto AppAquisBluet. Como produto resultante, apresentou-se as funcionalidades básicas e aquisições de dados em tempo real entre o software simulador e a aplicação desenvolvida intermediada pela comunicação Bluetooth, utilizando o protocolo RFCOMM (*Radio frequency communications*) criando uma porta serial virtual, para intermediar a comunicação entre dois dispositivos *Bluetooth* que estejam pareados. Haja vista, os dados trafegados poderem ser apresentados em "*text*", ou graficamente pela biblioteca *GraphView*, e armazenados no banco *SQLite* da própria aplicação para futuras análises.

Palavras-chave: Android. Google. Tecnologia Móvel. Bluetooth. Smartphone. ppAquisBluet.

ABSTRACT

The paper presents the development of AppAquisBluet application, using techniques and methods available on the Android platform. It is among several available in the most widespread market in the world for being a modern, flexible and open source platform, and has excellent cost/benefit ratio, ideal for the implementation of AppAquisBluet project. As resulting product, made up the basic features, and real-time data acquisition between the simulator software and the application developed brokered by Bluetooth communication, using the RFCOMM protocol (Radio frequency communications) creating a virtual serial port, to mediate communication between two Bluetooth devices that are paired. Considering that data traffic can be presented in "text", or graphically by GraphView library, and stored in the application's own SQLite database for future analysis.

Keywords: Android. Google. Tecnologia Móvel. Bluetooth. Smartphone. AppAquisBluet.

1 INTRODUÇÃO

A evolução da tecnologia móvel se deve aos investimentos e descobertas feitos no século passado. Diante disso, este trabalho procurou evidenciar quais foram os primeiros dispositivos móveis desenvolvidos na história, a evolução da comunicação sem fio e seus desdobramentos, a exemplo das criações das subáreas da computação móvel, redes de comunicação e as implementações de sistemas em dispositivos móveis, tendo como objetivo o desenvolvimento de uma aplicação móvel que seja capaz de realizar testes de aquisição de dados intermediada pela comunicação Bluetooth, direcionada a *smartphones* que são executados especificamente pelo sistema operacional Android.

De acordo com Lecheta (2013), mais de três bilhões de pessoas possuem um aparelho celular. Em percentual, isto corresponde aproximadamente mais da metade da população mundial. Com o crescimento considerável da utilização de celulares e usuários cada vez mais exigentes, houve a necessidade em agradar os dois segmentos de usuários, corporativos ou convencionais. Com isso, surgiram, ao longo dos anos, empresas especializadas em desenvolvimento de aplicativos móveis com plataformas específicas e tecnologias próprias.

Entre várias plataformas de desenvolvimentos de aplicativos móveis, a Apple iOS e Google Android detêm aproximadamente 91% do mercado móvel. Devido principalmente ao custo/benefício envolvendo as duas plataformas em questão, fez-se uso da plataforma Android para o desenvolvimento do projeto AppAquisBluet.

Usufruindo da tecnologia e ferramentas disponibilizadas na plataforma Android, implementou-se a estrutura do projeto AppAquisBluet, utilizando a IDE Eclipse. Como produto resultante das implementações feitas em código fonte, apresentou-se, o aplicativo AppAquisBluet e suas funcionalidades básicas, destes a interface do aplicativo, à execução de testes de aquisição de dados em tempo real, e posteriormente armazenou tal dados no banco *SQLite*, para futuras análises.

2 HISTÓRICO DA EVOLUÇÃO DA COMPUTAÇÃO MÓVEL

Diante de toda a evolução da computação móvel e descobertas de novas tecnologias que a influenciaram no decorrer do tempo, Mateus e Loureiro (1998) destacaram alguns pontos dessa trajetória: a descoberta de Hans Christian Oersted, no ano de 1820, em que a corrente elétrica produz um campo magnético; o primeiro sistema de comunicação desenvolvido - o telégrafo na metade do século XIX; a descoberta da radiotelegrafia, no final do século XIX, por Guglielmo Marconi; a invenção do segundo sistema de comunicação - o

telefone - por Alexander Graham Bell, e o computador, a terceira geração do sistema de comunicação.

A partir de várias implementações ocorridas ao longo dos anos, a comunicação sem fio evoluiu e se destacou, formando várias subáreas dentro da tecnologia digital, como: comunicação de celular, comunicação móvel, serviços de comunicação pessoal, comunicação via satélite, redes locais sem fios (MATEUS; LOUREIRO, 1998).

2.1 Sistemas operacionais para dispositivos móveis

Com as crescentes inovações tecnológicas, celulares, *smartphones* e dispositivos móveis ocupam cada vez mais tempo e espaço na vida das pessoas. Com tecnologia de processamento sempre mais veloz e com mais memória de armazenamento, o sucesso poderá ser garantido. E, o que antes era feito somente pelos Pcs, agora pode ser feito em qualquer lugar, hora do dia, direto na palma da mão com um *smartphone*.

A evolução no setor gerou investimentos de empresas como: Nokia, Apple, Microsoft, Samsung, Google, Intel, entre outras, que buscam garantir seu espaço na guerra da mobilidade com sistemas operacionais próprios para os *smartphones* (CAMARGO, 2010).

2.2 Plataforma de desenvolvimento: Google Android

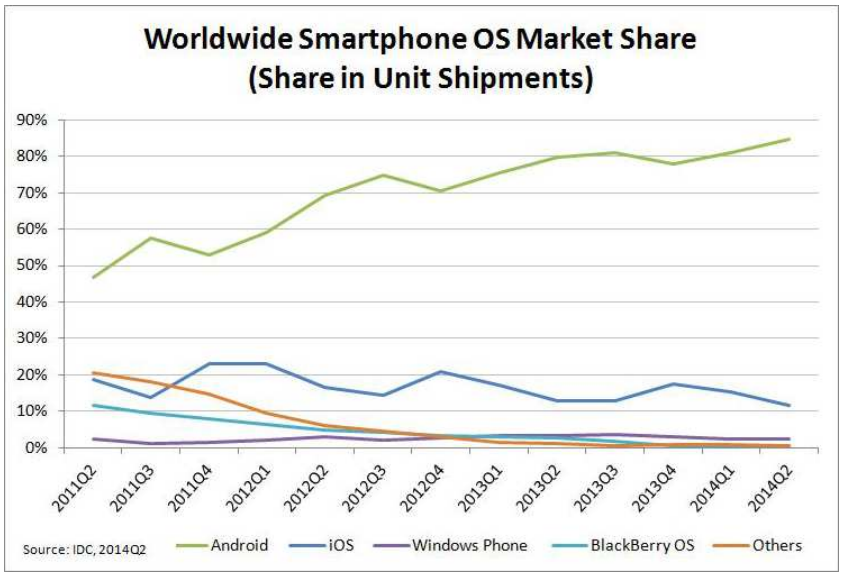
Observa-se, na literatura vigente, que o Android foi uma grande aposta da empresa Google, juntamente com a OHA (*Open Handset Alliance*), para garantir uma boa parte do mercado voltado à tecnologia móvel em crescente evolução. Com fortes investimentos da Google, o Android vem ganhando fãs em diversos setores onde a mobilidade está presente, seja com usuários convencionais, corporativos e desenvolvedores, pelo simples fato de ser uma plataforma leve e extremamente customizável.

Hoje, pode-se dizer que o Android não se limita apenas em *smartphones* ou *tablets*, pois é possível encontrar aplicações Android gerenciando vários tipos de dispositivos, como: televisores, microondas, geladeiras, carros e relógios, todos usufruindo da tecnologia disponibilizada pela Google com o sistema operacional Android (ANDROID, 2014).

Pelos investimentos proporcionados pela Google e a OHA, o Android se tornou nos últimos anos líder no mercado dos sistemas operacionais móveis, com mais de 800 mil aplicativos disponibilizados na Google Play Store, alcançando a marca de 1 milhão no ano 2013. Isso comprova que o Android teve uma excelente aceitação, detendo aproximadamente 75% do mercado de dispositivo móveis gerenciado pelo Android, em um mercado onde a tecnologia se evolui a cada instante, disponibilizando para o usuário final o que há de mais

moderno em tecnologia móvel e digital (QUERINO, 2013). Na Figura 1, pode-se constatar essa realidade apresentada por Querino (2013).

Figura 1 - Os sistemas operacionais mais utilizados na atualidade.

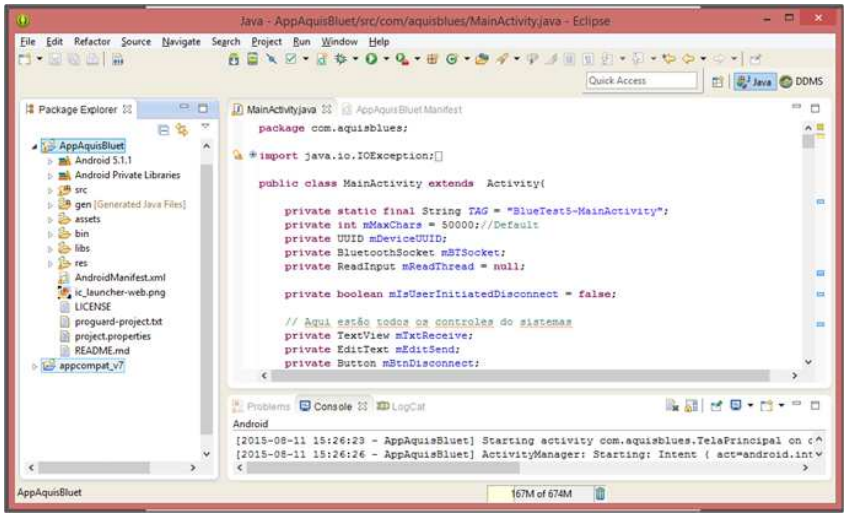


Fonte: (HAMANN, 2014).

3 MÉTODOS UTILIZADOS NO DESENVOLVIMENTO DO PROJETO APPAQUISBLUET NA IDE ECLIPSE

Observa-se, na criação de um projeto Android, utilizando a IDE Eclipse, como é o caso do AppAquisBluet, que seu núcleo é constituído de várias pastas, arquivos e bibliotecas. Para que o projeto funcione perfeitamente, alguns elementos são essenciais, conforme demonstrado na Figura 2. Isto é, a IDE Eclipse é composta pelas principais pastas que compõem o projeto em destaque.

Figura 2 - Estrutura do projeto na IDE.



Fonte: Elaborada pelo autor.

Na aba de *Packager Explorer* da IDE, têm-se as principais pastas que compõem o projeto AppAquisBluet, responsáveis pelo funcionamento do aplicativo, definidas por Lecheta (2013), como:

- **src** - a pasta que contém todos os arquivos fontes em java do projeto, como a classe MainActivity que foi criada pelo *wizard*;
- **gen** - contém todos os arquivos gerados automaticamente pela IDE. Nesta pasta encontra-se o arquivo responsável pela interface entre a codificação Java e XML, chamado R.java. Este arquivo é gerado pelo compilador que referencia todos os recursos do projeto, e o mesmo jamais deve ser alterado pelo desenvolvedor;
- **Android 5.1.1** - contém o arquivo android.jar, que possui todas as classes necessárias para uma aplicação Android;
- **assets** - contém arquivos HTML, arquivos texto, banco de dados, etc;
- **bin** - onde fica armazenado o arquivo de saída final do projeto, como o arquivo APK;
- **libs** - pasta que armazena as bibliotecas de terceiros, que estão em uso no projeto;
- **res** - possui algumas subpastas como: *layout*, menus, *values*, *drawable* e XML com configurações variadas, e serão automaticamente geradas na classe R, dependendo do conteúdo;
- **AndroidManifest.xml** - arquivo que descreve a natureza de uma aplicação e que contém expressamente cada um de seus componentes, além de ser

responsável por definir permissões, bibliotecas, versão do sistema e componentes de software que a aplicação necessitar;

- **project.properties** - arquivo de configuração gerado automaticamente, contém informações como a versão do pacote a ser utilizado para compilação.

A pasta *AndroidManifest.xml* é a base de uma aplicação Android, é obrigatória e deve ficar sempre na pasta raiz do projeto em desenvolvimento, uma vez que receberá toda a configuração necessária para executar aplicação. Ela armazenará o nome do pacote utilizado, a versão do SDK utilizado no projeto, as aplicações utilizadas, as *activities* que devem ser executadas, *inten-filter* para filtrar e executar a *activity*, permissões para usar algumas bibliotecas específicas e várias outras configurações, que serão armazenadas, dependendo do tipo de projeto (LECHETA, 2013).

3.1 Comunicação *Bluetooth* e API *Bluetooth*

A comunicação *Bluetooth* teve inicio no ano 1994, por parte da companhia *Ericsson*, utilizando uma tecnologia de sinais de rádio de baixo custo, que permitiu a comunicação entre telefones celulares e acessórios, deixando de lado os tradicionais cabos. Criou-se assim, o *MC-Link*, um sistema de rádio de curto alcance com uma implementação relativamente simples e barata. Logo, por ser uma tecnologia fácil e barata, despertou o interesse de outras empresas. Tanto que, no ano de 1998, foi criado o consórcio *Bluetooth* SIG (*Bluetooth Special Interest Group*), formado pelas companhias Ericsson, Intel, IBM, Toshiba e Nokia (BLUETOOTH, 2015).

A partir de então, o *Bluetooth* começou a virar realidade. Em julho de 1999, o grupo de trabalho IEEE 802.15WPAN (*Wireless Personal Area Network*) propôs a especificação *Bluetooth* versão 1.0 (THOMPSON; KUMAR; KLINE, 2008).

De acordo com Bluetooth (2015), com o passar dos anos foram disponibilizadas novas versões da tecnologia *Bluetooth* com o intuito de melhorar a taxa de frequência na comunicação e diminuir o consumo da bateria. Atualmente, a mesma encontra-se disponível no mercado na versão 4.1, lançada em 03 de dezembro de 2013. Contudo, para que seja possível atender aos mais variados tipos de dispositivos, o alcance máximo do *Bluetooth* foi dividido em três classes:

- Classe 1: potência máxima de 100 mW (*miliwatt*), alcance de até 100 metros;
- Classe 2: potência máxima de 2,5 mW, alcance de até 10 metros;

- Classe 3: potência máxima de 1 mW, alcance de até 1 metro.

É importante, no entanto, frisar que dispositivos de classes diferentes podem se comunicar sem qualquer problema, basta respeitar o limite daquele que possui um alcance menor. Diante do exposto, foi estabelecida a comunicação *Bluetooth* do aplicativo AppAquisBluet para trabalhar na Classe 2 com a transmissão de dados a faixa de 24 Mb/s.

A implementação das novas versões na tecnologia *Bluetooth*, no decorrer dos anos, possibilitou a comunicação com outros dispositivos, propiciando ao usuário dessa tecnologia a possibilidade de presenciar uma grande quantidade de cenários de comunicação *Bluetooth* possíveis.

3.1.1 *API Bluetooth para Android*

A importância da *API Bluetooth* para o projeto AppAquisBluet foi essencial, pois através dela implementou-se em código fonte todos os métodos necessários para que a comunicação via Bluetooth da aplicação funcionasse perfeitamente, tendo como objetivo, realizar aquisição de dados em tempo real.

Atualmente, as quatro principais tarefas, necessárias para se realizar uma comunicação via *Bluetooth* são: configurar o *Bluetooth*, encontrar dispositivos que estejam emparelhados ou disponíveis na área, conectar os dispositivos e transferir dados entre os mesmo. Mas, para que essa ação seja concretizada, precisa-se iniciar um *link* de comunicação, utilizando os *Sockets Bluetooth*, tornando possível a ação de transmitir e receber dados entre os dispositivos.

Utilizando a API *Bluetooth* do Android, o dispositivo *Bluetooth* local é controlado através da classe *BluetoothAdapter*, que representa o dispositivo Android em que a aplicação está sendo executada. Dessa forma, para acessar o *adaptador Bluetooth* padrão da API, é necessário utilizar o método *getDefaultAdapter()*. Contudo, para iniciar a busca por dispositivos *Bluetooth*, deve-se incluir as permissões do *Bluetooth* no *AndroidManifest.xml* e modificar as propriedades do dispositivo local, onde a permissão de *BLUETOOTH_ADMIN* deve ser concedida (LECHETA, 2013). Na Figura 4, apresentam-se as linhas de códigos a serem adicionadas no arquivo *AndroidManifest.xml*, para que seja concedida a comunicação entre os dispositivos.

Figura 4 - Permissão *Bluetooth no AndroidManifest*

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

Fonte: AppAquisBluet desenvolvido pelo autor.

Segundo Lecheta (2013), o *adaptador Bluetooth* oferece métodos para leitura e configuração das propriedades do hardware *Bluetooth*, em que o adaptador somente poderá ler ou configurar, se estiver habilitado no dispositivo. Caso não esteja habilitado, o método *getDefaultAdapter*() retorna *null*. Este teste é feito no *onCreate()* da *activity* principal do projeto, para saber se o dispositivo está habilitado com esta função. Na Figura 5, pode-se verificar tal código que faz a verificação se o *Bluetooth* está ativo no dispositivo.

Figura 5- Código para verificar se o *Bluetooth* está ativo.

```
mBTAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBTAdapter == null) {
    Toast.makeText(getApplicationContext(), "Bluetooth não encontrado", Toast.LENGTH_SHORT).show();
```

Fonte: AppAquisBluet desenvolvido pelo autor.

Umas das ações que são corriqueiras por parte dos usuários é manter o *Bluetooth* desabilitado até utilizá-lo, para conservar a carga da bateria. Mas, para habilitar o adaptador *Bluetooth*, pode-se usar a constante *BluetoothAdapter.ACTION_REQUEST_ENABLE* com o método *startActivityForResult()*, contexto em que aparecerá uma tela pedindo que o usuário habilite o *Bluetooth*. Na Figura 6, apresenta-se o código utilizado para habilitar o *Bluetooth* no dispositivo.

Figura 6 - Código de solicitação da ativação do *Bluetooth*.

```
else if (!mBTAdapter.isEnabled()) {
      Intent enableBT = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
      startActivityForResult(enableBT, BT_ENABLE_REQUEST);
  } else {
      new SearchDevices().execute();
  }
```

Fonte: AppAquisBluet desenvolvido pelo autor.

Depois de concedida a permissão do *BLUETOOTH_ADMIN* incluída no *AndroidManifest.xml*, é possível habilitar e desabilitar o *Bluetooth* do dispositivo, usando os métodos *enable e disable*. A descrição das classes dessas API, utilizadas para configurar o *Bluetooth*, procurar por dispositivos, conectar-se aos dispositivos pareados e transferir dados entre os dispositivos, é definida por Lecheta (2013) como:

- ***BluetoothAdapter*** - representa o adaptador *Bluetooth*, em que, através dessa classe, é possível procurar por outros dispositivos e se conectar a eles;
- ***BluetoothDevice*** - representa o outro dispositivo *Bluetooth*, que pode solicitar ao dispositivo remoto uma conexão ou informações do dispositivo;
- ***BluetoothSocket*** - representa a interface para um soquete *Bluetooth*;
- ***BluetoothServerSocket*** - representa um servidor aberto para atender a requisições de outros dispositivos, função essa que funciona como um *host*.

3.2 Estabelecendo canais RFCOMM na comunicação *Bluetooth*

O ponto fundamental que envolve a comunicação Bluetooth é a RFCOMM (*Radio frequency communications*), pois é através dela que os dispositivos se comunicam transmitindo informações. Para o projeto AppAquisBluet esse foi o protocolo fundamental, pois procurávamos algo que fosse barato, seguro e eficaz. Os testes realizados provaram que o protocolo funcionou perfeitamente, apresentado os resultados conforme eram enviados pelo hardware e recebidos pela aplicação conforme foram implementados e tratados.

Segundo Bluetooth (2015), as API's do *Bluetooth* para Android baseiam-se no protocolo de comunicação RFCOMM (*Radio frequency communications*), simulando um protocolo de substituição de cabo, usado para criar uma porta serial virtual, tornando um mecanismo para abrir soquetes de comunicação entre dois dispositivos *Bluetooth* que estejam pareados. Contudo, para estabelecer essa comunicação *Bluetooth* RFCOMM bidirecional, o programador pode utilizar as classes: *BluetoothServirSocket* e *BluetoothSocket*, em que a primeira será implementada em um dispositivo se passando por servidor que atende e aceita pedidos de conexão, e a segunda será implementada como um cliente. Após estabelecida a conexão entre os dispositivos, será feita a transferência de dados por intermédio da classe *BluetoothSocket*.

Lecheta (2013), relata que, tanto a classe *BlutoothServirSocket* como a *BluetoothSocket* utilizam-se do UUID (*Universally Unique Identifier*) para se comunicar, com um formato padronizado para ID *string* de 128 bits usado para identificar de forma única a informação. A vantagem de se utilizar o UUID é que este é grande o bastante para que se possa selecioná-lo aleatoriamente sem que haja colisões de identificadores.

O método *listenUsingRfcommWithServiceRecord (String, UUID)* é usado para deixar o dispositivo "esperando" por pedidos de conexão, em que os parâmetros são o nome que identifica o servidor e o identificador universal que, obrigatoriamente, terá que ser confirmada pelo cliente para que possa se conectar ao servidor. Cria-se assim, um RFCOMM *BluetoothSocket*, pronto para iniciar uma conexão de saída segura para este dispositivo remoto, usando *lookup* SDP (*Service Discovery Protocol*), de UUID utilizando o método *createRfcommSocketToServiceRecord (UUID).* Se a conexão for concluída com sucesso, o método *accept()* retornará um *BluetoothSocket* e este pode ser usado para transferência de dados entre os dispositivos (LECHETA, 2013).

Depois de conectados, haverá apenas um *BluetoothSocket* para o dispositivo cliente e o servidor. Contudo, pode-se afirmar que o conceito estabelecido entre o cliente e servidor só

foi utilizado para exemplificar o processo de comunicação entre os dispositivos até a conexão ser estabelecida. A partir deste momento, não existe distinção entre os dispositivos, pode-se enviar e receber dados através do *BluetoothSocket*, pois a transferência de dados entre os dispositivos é tratado pelas classes Java *OutputStream* e *InputStream*, obtidas do *BluetoothSocket*, através dos métodos *getOutputStream* e *getInputStream* (LECHETA, 2013).

3.3 Biblioteca *GraphView*

Entre muitas bibliotecas gráficas disponibilizadas para implementação gráfica, utilizou-se a *GraphView* no projeto AppAquisBluet, por ser uma biblioteca destinada para aplicações Android com objetivo de criar programaticamente diagramas flexíveis e de boa aparência, já que é fácil de entender, integrar e personalizá-los. Visto que, é possível alterar os dados do gráfico em tempo de execução, ou seja, em tempo real. Para fazer isso, disponibilizam-se dois métodos importantes nas subclasses da série:

- ***resetData*** - este método redefine todos os dados, para que os dados atuais sejam substituídos pelo novo;
- ***appendData*** - esse métodos acrescentam um único conjunto com os dados atuais de dados. Há também a função *"scrollToEnd"*, que irá percorrer a *GraphView* automaticamente para o último valor de X (GRAPHVIEW, 2015). Essas são algumas funções que fizeram com que utilizássemos a biblioteca no projeto, por serem simples e eficazes como mostrou os resultados em testes realizados.

3.4 Banco de dados

O banco de dados é definido como um sistema computacional que armazena informações por meio eletrônico para futuras consultas, alterações ou exclusão desses dados, caso seja necessário (MEDNIEKS; DORNIN; NAKAMURA, 2012).

Com base na definição de Mednieks, Dornin e Nakamura (2012), implementou-se no aplicativo AppAquisBluet a função de banco de dados, haja vista, que a aplicação tem por objetivo principal realizar aquisições de dados em tempo real, utilizando a comunicação Bluetooth, com isso é de suma importância que esses dados fossem armazenados em banco para futuras análises.

Para armazenar tais dados, utilizou-se o banco *SQLite*, pois o mesmo pode ser implementado na própria aplicação sem que haja a necessidade de se preocupar com sistema de gerenciador de dados, pois o objetivo do projeto é realizar aquisição de dados em tempo real utilizando a comunicação Bluetooth, ou seja, descartando qualquer tipo de comunicação intermediada pela internet na aquisição dos dados.

3.4.1 *Banco de dados SQLite*

O *SQLite* é definido como um banco de dados estável, de código aberto, amplamente utilizado em dispositivos com limitações no volume de memória ou armazenamento de dados. Seu diferencial consiste na ausência de um processo executado no sistema operacional exclusivo para o servidor. Isso acontece porque o *SQLite* escreve diretamente em arquivos comuns no sistema de armazenamento em disco (SQLITE, 2015).

Uma aplicação Android se conecta com o banco *SQLite* através de uma interface denominada "*DbHelper"*. Para criar tal interface, é necessário que se crie uma classe auxiliar que realize a conexão através da utilização de outra classe contida no *framework* do sistema Android, denominada de "*SQLiteOpenHelper"*, que retorna uma instância do tipo *"SQLiteDatabase"* (MEDNIEKS; DORNIN; NAKAMURA, 2012).

As operações básicas de inserção, alteração, remoção e execução de consultas personalizadas estão presentes como parte da plataforma. Elas podem ser acessadas através de funções contidas no *framework* do sistema operacional. As funções, quando executadas, retornam um conjunto de colunas para um ponteiro do tipo "cursor" contendo as informações solicitadas no código, no qual é possível acessar uma por vez por meio de comandos determinados por meio de código (SMITH; FRIESEN, 2012).

4 RESULTADO DOS MÉTODOS UTILIZADOS E AQUISIÇÃO DE DADOS VIA BLUETOOTH

Neste tópico, são apresentados os resultados dos métodos utilizados no desenvolvimento do AppAquisBluet, tendo como objetivo apresentar a interface do aplicativo e suas funcionalidades. Executaram-se testes de aquisição de dados e plotou-se os resultados dos dados no gráfico, utilizando a biblioteca *GraphView* e armazenados no banco *SQLite* para futuras análises.

4.1 AppAquisBluet

O projeto AppAquisBluet ficou constituído de várias pastas, arquivos e bibliotecas que, juntas, formam a estrutura do projeto desenvolvido na IDE Eclipse. Implementou-se suas *activity, intent, service, broadcast receiver*, resultando na tela principal do aplicativo e suas funcionalidades.

Quando o aplicativo é executado, o usuário se depara com a *activity* inicial (tela principal), composta por três botões, em que cada um direciona para uma *activity* diferente com funcionalidade específica. Contudo, para que fosse possível tal funcionalidade, foi utilizada técnica para inserir os botões, criando um *LinearLayout* sem componentes, contendo

apenas os botões com “texto”, utilizando a função *onClickListener* no próprio *LinearLayout*. Assim, pode-se observar, a seguir, a função de cada botão em destaque:

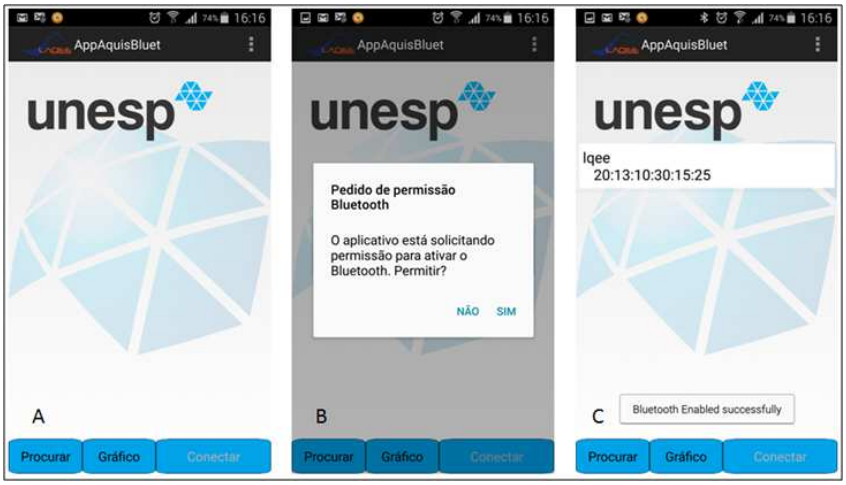
- Acionando em **Procurar** - o usuário ativará a função de procurar por dispositivos que estejam pareados e dentro do raio de alcance do *Bluetooth*;
- Acionando em **Gráfico** - o usuário será direcionado para a tela onde será plotado os resultados de aquisição de dados em gráfico;
- Acionando em **Conectar** - o usuário será direcionado à tela de aquisição de dados em tempo real, onde o mesmo poderá inicializar a aplicação propriamente dita.

4.2 Resultado do pareamento entre o dispositivo e o módulo *Bluetooth*

O aplicativo AppAquisBluet só funcionará se o *Bluetooth* do celular estiver ativo. Portanto, a primeira verificação que a *activity* faz, ao ser executada quando aciona-se o botão “Procurar”, é saber se o dispositivo Android está com o recurso do *Bluetooth* habilitado. Caso não esteja, uma *activity* central será inicializada, solicitando a ativação do *Bluetooth*. Essa solicitação é automática e nativa de dispositivos Android.

Após permitir a ativação do *Bluetooth*, o usuário precisará conectar o celular ao módulo *Bluetooth* e iniciar o processo de aquisição de dados. Para isso, é preciso clicar no botão “Procurar” do aplicativo AppAquisBluet e, então, aceitar o pedido de permissão *Bluetooth*. A aplicação irá exibir a *DeviceListActivity*, que permite fazer uma procura por dispositivos *Bluetooth* disponíveis e selecionar à qual irá se conectar. O módulo *Bluetooth* que será conectado está representado como “lqee (20:13:10:30:15:25)” como mostrado na Figura 7, na letra “C”.

Figura 7 - Tela Principal do AppAquisBluet.



Fonte: Elaborado pelo autor.

4.3 Tela de aquisição de dados em tempo real

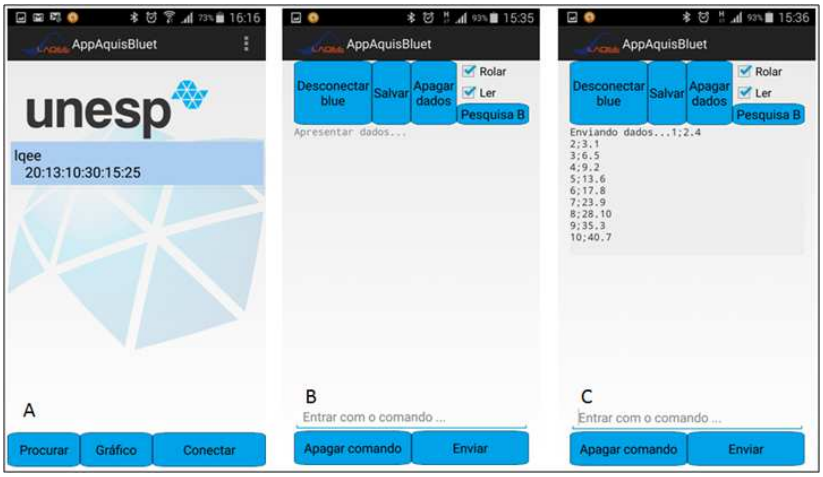
Utilizando-se da API *Bluetooth* disponibilizada para o Android, juntamente com seus métodos e classes, criou-se um canal de comunicação entre o dispositivo e o módulo *Bluetooth*, realizando assim, aquisição de dados em tempo real, utilizando a comunicação RFCOMM (*Radio frequency communications*), simulando um protocolo de substituição de cabo usado para criar uma porta serial virtual, se tornando um mecanismo para abrir soquetes de comunicação entre dois dispositivos *Bluetooths* que estejam pareados para receber e enviar dados.

Na Figura 8, apresenta-se a tela de aquisição de dados, contendo algumas funções especificas, as quais podem ser observadas na tela representada pela letra "B" contendo alguns botões:

- **Desconectar bluet** - tem a função de encerrar o canal de comunicação entre o dispositivo e o módulo *Bluetooth*;
- **Salvar** - tem a função de salvar os dados recebidos no banco *SQLite* da aplicação;
- **Apagar dados** - tem a função de deletar todos os dados recebidos no campo "apresentar dados";
- O *checkboxe* **Rolar** - tem a função de limitar a visão dos dados na tela da aplicação e o *checkboxe* **Ler,** tem a função de cancelar a leitura de dados no visor da aplicação;
- **Pesquisa B** - tem a função de direcionar o usuário para uma nova janela na qual poderá consultar dados armazenados em banco;
- Na parte inferior da aplicação, apresenta-se, no visor, o campo onde será digitado o comando, e dois botões, um para envio de comandos para liberação de dado e o outro para apagar comando digitado.

Na tela representada pela letra "C", da Figura 8, pode-se observar no visor da aplicação uma matriz três por dez, resultado da aquisição de dados em tempo real intermediada pela comunicação entre o módulo *Bluetooth* e o dispositivo móvel. Sendo que os dados recebidos do módulo *Bluetooth*, foram armazenados na "String strInput" e apresentado no visor da aplicação.

Figura 8 - Tela de aquisição de dados.



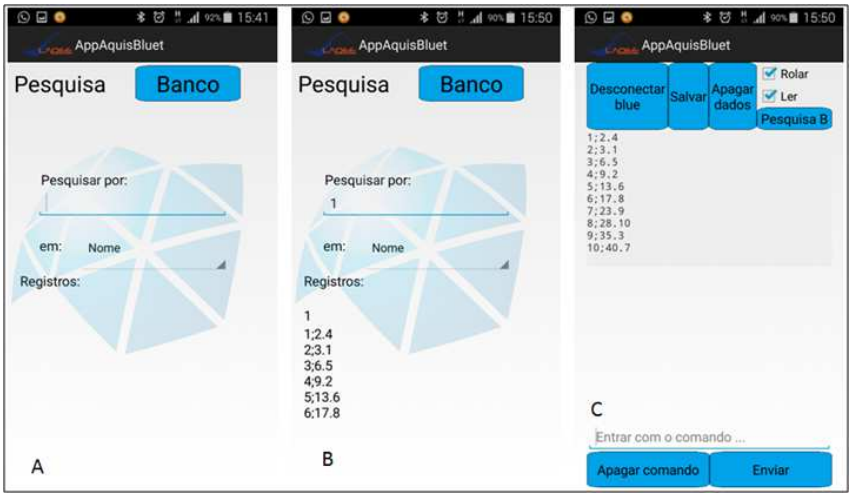
Fonte: Elaborado pelo autor.

4.4 Pesquisando dados no banco

Para realizar uma pesquisa de dados em banco, foram utilizadas técnicas e métodos da linguagem *SQL* implementou-se, em seu núcleo, os elementos básicos e modelos de dados relacionados ao armazenamento de dados. Para o projeto em destaque, utilizou-se o banco *SQLite*, por ser recomendado para aplicações móveis e com funcionalidades que torna os dados seguros e confiáveis. Como resultado da implementação apresenta-se na Figura 9, três telas:

- Tela **A** - apresenta o campo onde o usuário irá digitar o número do “id”, representado em banco através da chave primária na primeira coluna da tabela "dados". Logo abaixo, tem-se o registro dos dados, do tipo “text”, armazenados na segunda coluna da tabela “dados” referente à pesquisa. Em seguida, o usuário irá clicar no botão “Banco” e os dados serão apresentados no campo de registros, conforme a busca realizada no “id”;
- Tela **B** - pode-se observar o resultado da pesquisa realizada no identificador 1 e seus respectivos dados;
- Tela **C** - apresenta apenas uma confirmação da entrada de dados em tempo real no visor da aplicação, sendo confirmada a mesma matriz no registro do banco de dados pelo identificador 1.

Figura 9 - Tela de pesquisa no banco.



Fonte: Elaborado pelo autor.

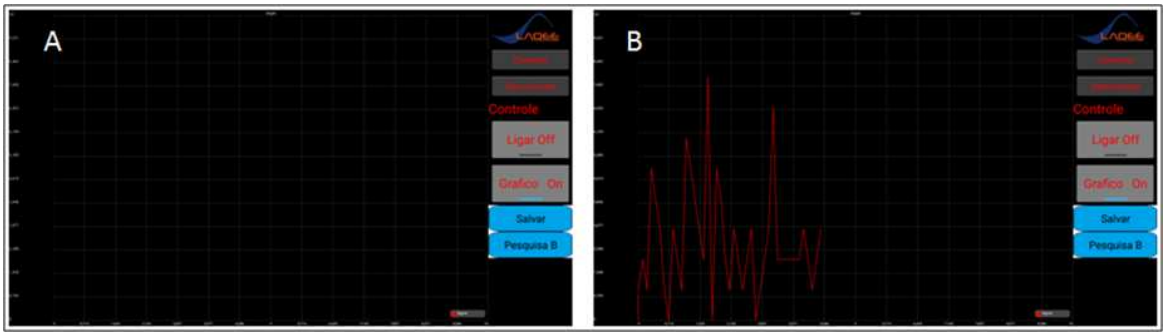
4.4.1 *Pesquisando dados gráficos em banco*

Na tela principal do AppAquisBluet na Figura 7, tem-se o botão "Gráfico" que direcionará o usuário para uma nova janela. Contexto em que será realizada aquisição de dados em tempo real e o resultado será plotado graficamente, utilizando a biblioteca *GraphView*. Como resultado das implementações utilizando métodos, classes e bibliotecas da programação Android, apresenta-se na Figura 10, a tela "A", em que pode-se observar o eixo "Y" com tamanho de 0 a 10 e o eixo "X" em uma crescente de 0 a 10. O gráfico pode ser alterado pelo botão "Gráfico On e Off", além de outras configurações que podem ser observadas na mesma tela.

- O botão de **conectar** e **desconectar** a aplicação ao módulo *Bluetooth*;
- O botão de Ligar **Off** e **On**, para liberar a entrada de dados para plotar no eixo Y e X;
- O botão de **Gráfico On** e **Off**, para limitar a entrada de dados;
- O botão **Salvar** irá salvar os dados recebidos em banco para futuras análises;
- O botão **Pesquisa B**, que direciona o usuário para uma nova janela, na qual será realizada a pesquisa no banco *SQLite*.

Na tela representa pela letra "B", na Figura 10, pode-se constatar a aquisição de dado plotada no eixo Y e X com os respectivos valores V = (1; 2; 1; 5; 4; 3; 1; 0; 3; 2; 1; 6; 5; 4; 3; 2; 8; 0; 5; 4; 2; 1; 3; 2; 1; 2; 3; 0; 1; 2; 3; 7; 2; 2; 2; 2; 2; 2; 3; 2; 1; 2; 3). Sendo "**V**", os valores recebidos aleatoriamente, transmitido pelo módulo *Bluetooth*, armazenados em uma "String strIncom" na aplicação, sendo plotado um caractere por vez no eixo Y com distância continua de 0,1 no eixo X utilizando "**;**"como separador dos dados recebidos.

Figura 10 - Aquisição de dados.

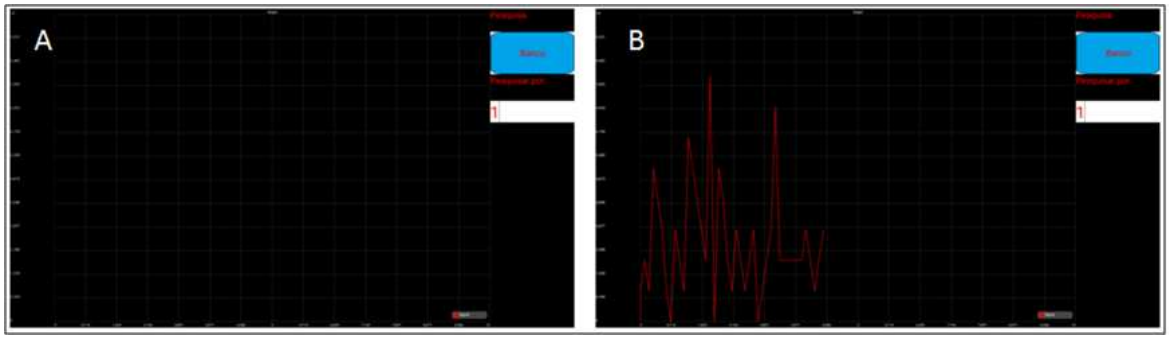


Fonte: Elaborado pelo autor.

Realizando uma busca em banco, pode-se observar a aquisição de dados demonstrada na Figura 10, sendo apresentada na Figura 11.

Na tela, “A” da Figura 11, visualiza-se a tela banco, na qual será plotado o resultado da pesquisa realizada no identificador “1” representado em banco através da chave primária na primeira coluna da tabela “gráfico” e o valor do tipo “**text**” armazenado na segunda coluna. Preenchendo o campo do identificador e acionando-se o botão “Banco” o resultado da aquisição armazenada em banco no “id 1” será plotado na tela “B” da Figura 11.

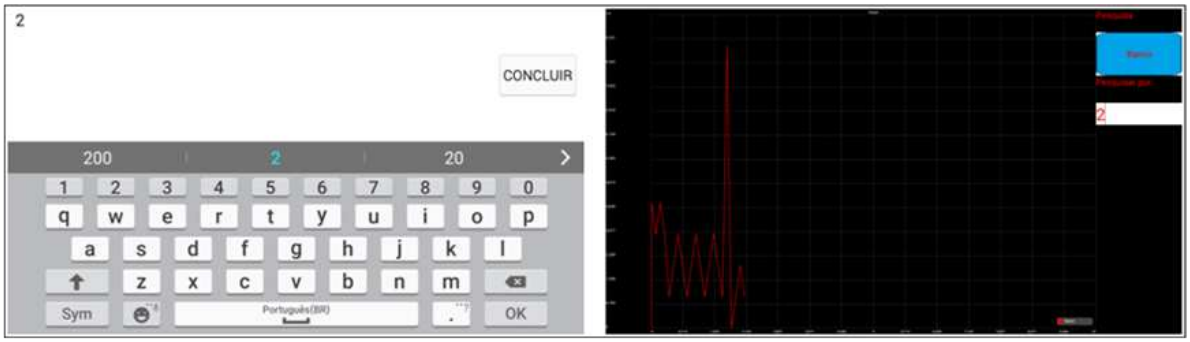
Figura 11- Busca no banco de dados.



Fonte: Elaborado pelo autor.

Para realizar uma nova pesquisa de dados gráficos em banco, o usuário terá que digitar o número de identificação do “id” desejado e clicar em “OK”. Em seguida, será direcionado à tela “Banco” com o “id” desejado já preenchido e, acionado o botão “Banco” o resultado da pesquisa realizada pelo identificador escolhido foi plotado no eixo “Y” e “X”. Isto pode ser observado na Figura 12, com os respectivos valores V = (4; 3; 4; 3; 1; 2; 3; 2; 1; 2; 3; 2; 1; 2; 3; 2; 1; 9; 0; 1; 2; 1). Sendo “**V**”os valores plotados no gráfico e armazenados em banco através da “String dadobanco” e posteriormente plotado um caractere por vez no eixo Y com distância continua de 0,1 no eixo X, utilizando “**;** ” como separador de dados.

Figura 12 - Resultado da pesquisa em banco no “id 2”.



Fonte: Elaborado pelo autor.

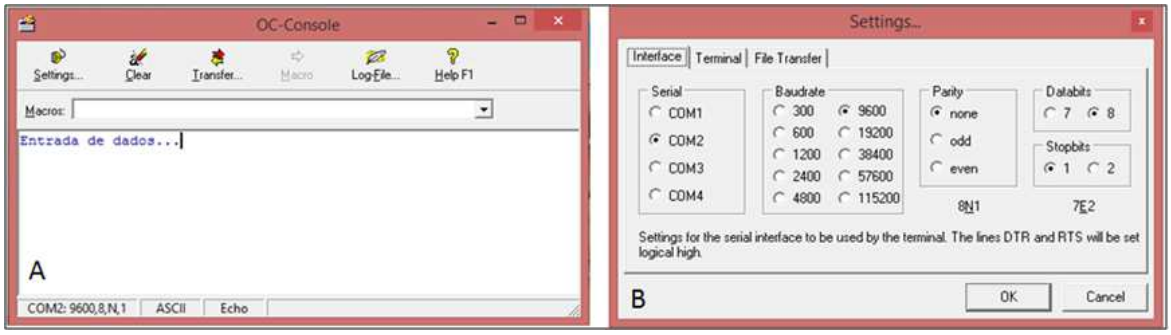
4.5 Tecnologias utilizadas na aquisição de dados

Dentre as tecnologias utilizadas, destacam-se, o software OC-Console, os hardwares *Ultrabook inspiron 14Z*, Samsung Galaxy S5 e o módulo *Bluetooth* HC-06.

4.5.1 *Software OC-Console*

Fez-se uso da versão *Release 2.5*, tendo a Castlesoft como detentora dos direitos autorais do software. Tal versão é definida como um programa de terminal, cuja tarefa é simular uma porta serial para receber e enviar dados, contendo funções básicas de um programa de terminal com frequência suficiente para satisfazer completamente o desenvolvimento de sistemas embarcados, como: taxa de transmissão selecionável, porta COM selecionáveis e função de transferência de pastas, como pode-se observar na Figura 13.

Figura 13- Interface do programa de terminal OC-Console.



Fonte: Elaborado pelo autor.

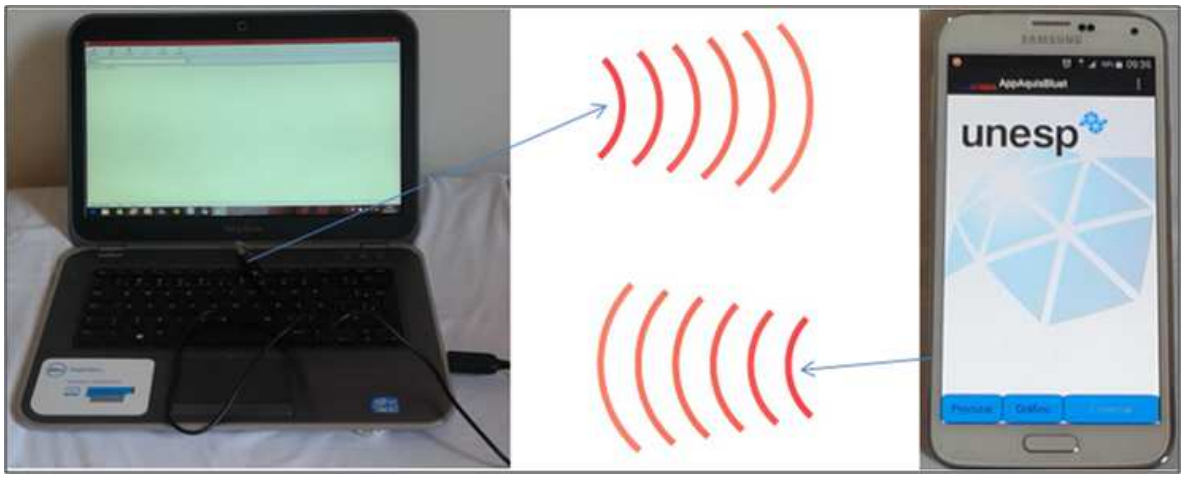
Na tela “B”, apresenta-se a configuração da interface serial, na qual será selecionada uma das quatro portas seriais do tipo “COM”, e escolhidos os parâmetros para a taxa de transmissão, os *databits, os stopbits* e a paridade.

4.5.2 Estabelecendo a comunicação

Na Figura 14 apresenta-se o computador, o *smartphone* e o módulo *Bluetooth*, utilizados, para simular um sistema de aquisição de dados com transmissão via *Bluetooth*, a fim de otimizar os testes em software, sem preocupação com o hardware. Assim, a

comunicação entre o programa de terminal OC-Console instalado no *Ultrabook* e o AppAquisBluet instalado no *smartphone* foram concretizadas em sua totalidade. Quanto à configuração dos dispositivos envolvidos na aquisição, temos: o *Ultrabook*, com sistema operacional de 64 bits, Windows 8.1 *Single Language*, processador Intel (R) Core (TM) i5-3337U CPU 1.80GHz, Memória (RAM) de 4 GB e o *smartphone*: Samsung S5, modelo SM-G900M, versão do Android 5.0 e 16 GB de armazenamento.

Figura 14 - Comunicação *Bluetooth* utilizando parâmetros da classe 2.



Fonte: Elaborado pelo autor.

4.5.3 Módulo Bluetooth HC-06

O módulo *Bluetooth* HC-06 (*Bluetooth* versão 2.0) transmite e recebem dados através das portas seriais TXD (Transmissão de Dados) e RDX (Recepção de Dados), utilizando taxas de 1.200, 2.400, 4.800, 9.600, 19.200 e 38.400 bps. Logo, a taxa definida para realizar os testes relacionados a este projeto foi a 9.600 bps. Assim, pode-se ressaltar que a alimentação do módulo *Bluetooth* pode utilizar a tensão entre 3,6 a 6 volts, possui antena embutida e atua somente em modo *slave* (escravo).

Para que fosse possível estabelecer a comunicação entre o OC-Console e o *smartphone*, realizou-se uma adaptação USB no módulo *Bluetooth* para que fosse possível realizar a transmissão de dados utilizando uma porta serial, como mostra a Figura 15.

Figura 15 - Módulo *Bluetooth* HC-06 utilizado no projeto.

Fonte: Elaborado pelo autor.

5 CONCLUSÕES

Em pleno século XXI, a tecnologia digital mantém uma evolução constante e muitas coisas são realizadas com o auxílio de computadores ou dispositivos móveis. Diante deste cenário, a linha de pesquisa deste estudo voltou-se para a tecnologia móvel. Assim, após feitas várias análises em materiais disponibilizados na literatura vigente, decidiu-se adotar a plataforma de desenvolvimento Android por ser uma plataforma, onde o custo/beneficio foi mais aceitável para o projeto, quando comparada às demais plataformas, a exemplo da plataforma Apple iOS.

Escolher tal caminho ficou mais simples, visto que a Google, juntamente com seus parceiros, disponibilizou para o desenvolvedor, seja ele corporativo ou convencional, uma plataforma moderna, flexível e *open-source*, baseada no *kernel do Linux*, além de ser o sistema mais disseminado no mundo, para variadas marcas de *smartphones*.

Como produto resultante do projeto AppAquisBluet, desenvolvida na IDE eclipse, utilizando a tecnologia de programação Android, *API bluetooth*, comunicação serial simulada por RFCOMM, implementação gráfica pela biblioteca *GraphView* e armazenamento de dados através dos métodos de banco de dados *SQLite*, chegou-se ao resultado de aquisições de dados preliminares utilizando o *notebook* para simular o sistema real, como pode ser observado no tópico 4 desde trabalho.

Quanto aos dados, enviados pelo programa de terminal OC-Console, foram recebido e apresentado no visor da aplicação conforme foram implementados e tratados, sem que houvesse alguma alteração em suas características. As funções estabelecidas no aplicativo AppAquisBluet, funcionaram perfeitamente como era esperado, respeitando todos os parâmetros estabelecidos e implementados em código fonte. Seja na navegação de telas, aquisição de dados no formato “texto” ou graficamente e o armazenamento desses dados em banco, para futuras analises.

REFERÊNCIAS

ANDROID, D. **Build for a multi-screen world.** 2014. Disponível em: <http://developer.android.com/index.html>. Acesso em: 23 jul. 2014.

BLUETOOTH. **A look at the basics of bluetooth technology.** 2015. Disponível em: <http://www.bluetooth.com/Pages/Basics.aspx>. Acesso em: 14 maio. 2015.

CAMARGO, C. **Sistemas operacionais móveis***:* qual a diferença? 2010. Disponível em: <http://www.tecmundo.com.br/samsung/3702-sistemas-operacionais-moveis-qual-a-diferenca-.htm>. Acesso em: 21 mar. 2014.

GRAPHVIEW, A. **Key features**. 2015. Disponível em: <http://www.android-graphview.org/>. Acesso em: 23 maio. 2015.

HAMANN, R. **O líder absoluto no mercado**. 2014. Disponível em: <http://www.tecmundo.com.br/sistema-operacional-/60596-ios-android-windows-phone-numeros-gigantes-comparados-infografico.htm>.Acesso em: 23 maio. 2015.

LECHETA, R. R. **Google Android***:* aprenda a criar aplicações para dispositivos móveis com o Android SDK. São Paulo: Novatec Editora, 2013.

MATEUS, G. R.; LOUREIRO, A. A. F. **Introdução à computação móvel**. Rio de Janeiro: DCC/IM, COPPE/UFRJ, 1998.

MEDNIEKS, Z.; DORNIN, G. L.; NAKAMURA, M. **Programando o Android**. São Paulo: Novatec, 2012.

QUERINO, L. C. F. **Desenvolvendo seu primeiro aplicativo Android***:* entre de cabeça no mundo dos aplicativos móveis, criando e publicando seu próprio programa para o sistema líder do mercado! São Paulo: Novatec Editora, 2013.

SMITH, D.; FRIESEN, J. **Receita Android** *:*uma abordagem para resolução de problemas. Rio de Janeira: Ciência Moderna., 2012.

SQLITE. **About SQLite**. 2015. Disponível em: <http://http://www.sqlite.org/about.html>. Acesso em: 25 mar. 2015.

THOMPSON, T. J.; KUMAR, C. B.; KLINE, P. J. **Bluetooth application programming with the Java APIs essentials edition**. San Francisco, California: Morgan Kaufmann, 2008.

CONFLITO DE INTERESSES

Os autores declaram não haver conflito de interesses.

AUTOR PARA CORRESPONDÊNCIA

Luis Carlos Origa de Oliveira
Universidade Estadual Paulista - Faculdade de Engenharia de Ilha Solteira - FEIS/UNESP
15385-000, Ilha Solteira-SP, Brasil
origa@dee.feis.unesp.br

Submetido em 24/09/2015
Aceito em 22/10/2015