Sistema Operacional Android

Cainã dos Santos Oliveira, Gustavo da Silva Bernardes, Luiz Carlos Ferreira Júnior & Uéslei Emídio Pivoto.

Abstract **– In a large rise in the market, the operating system for mobile devices Android reached, accord recent research, approximately 48% of smartphone's global market. This platform is headed by Google company and it's source code is open, allowing developers to make their own applications.**

Index Terms **– Operating System, Mobile devices, Google.**

Resumo **– Em grande ascensão no mercado, o sistema operacional para dispositivos móveis Android, conquistou segundo recentes pesquisas, cerca de 48% do mercado global de smartphones. Esta plataforma é de responsabilidade da empresa Google e tem seu código aberto, permitindo que desenvolvedores façam seus próprios aplicativos.**

Palavras chave **- Sistema operacional, Dispositivos móveis, Google.**

1 INTRODUÇÃO

O propósito deste documento é apresentar o trabalho de conclusão de curso dos alunos do décimo período de Engenharia da Computação referente ao sistema operacional de dispositivos móveis Android. Serão abordadas as características principais deste sistema operacional, bem como um estudo de caso onde foi desenvolvido um aplicativo para gerenciar os gastos pessoais através de um telefone celular com este sistema operacional.

2 HISTÓRIA E EVOLUÇÃO DO ANDROID

O Android é um sistema operacional para dispositivos móveis que roda sobre o núcleo Linux. Foi inicialmente desenvolvido pela Google e posteriormente pelo grupo Open Handset Alliance (OHA), mas a Google é a responsável pela gerência do produto e engenharia de processos.

A OHA foi criada com a intenção de padronizar uma plataforma de código aberto e livre para celulares. Esse grupo possuía 47 integrantes de peso, dentre eles, grandes multinacionais do mercado como HTC, LG, Motorola, Samsung, Sony, Ericsson, Toshiba, Sprint, Nextel, China Mobile, T-Mobile, Asus, Intel, Garmim e muitas outras.

Em julho de 2005 a Google adquiriu a Android Inc., uma pequena empresa em Palo Alto, California, USA. Conduzido por Andy Rubin, foi desenvolvida uma plataforma de telefone móvel baseado em Linux, com o objetivo de ser uma plataforma flexível, aberta e de fácil migração para os fabricantes.

O Sistema surgiu como sendo uma das grandes apostas do Google para ocupar espaço no mercado de smartphones que só vem crescendo no decorrer dos anos, o primeiro smartphone lançado usando esta plataforma foi o T-Mobile desenvolvido pela HTC, que começou a ser comercializados nos Estados Unidos no dia 22 de outubro de 2008 por US$ 179, e como esperado, causou um grande impacto no mercado e superou as expectativas de vendas.

O primeiro tablet comercialmente disponível a rodar o sistema operacional Android 3.0 foi o Motorola Xoom, lançado a 24 de fevereiro de 2011 nos Estados Unidos.

Versões do Android:

- 1.5: Cupcake (Abril de 2009)
- 1.6: Donut (Setembro de 2009)
- 2.1: Eclair (Janeiro de 2010)
- 2.2: FroYo (Maio de 2010)
- 2.3: Gingerbread (versão lançada em 6 de dezembro de 2010)
- 3.0-3.2: Honeycomb (Lançada especialmente para tablets em Janeiro de 2011)
- 4.0 Ice Cream Sandwich - Lançamento previsto para o quarto trimestre de 2011, o número da versão ainda não foi oficialmente revelado. (A versão que promete acabar com a fragmentação, ou seja, será para smartphones e tablets de todos os tipos).

3 ARQUITETURA ANDROID

O Android foi desenvolvido inicialmente para smartphones, hoje é usado em diversas outras aplicações como tablets e até relógios. Apesar de ser baseado no kernel do Linux, existe pouca coisa em comum com distribuições Linux convencionais (embarcadas ou não). À grosso modo, o Android é uma máquina virtual Java rodando sobre o kernel do Linux, dando suporte para o desenvolvimento de aplicações Java através de um conjunto de bibliotecas e serviços.

De acordo com a Fig. 1, sua arquitetura tem quatro camadas:

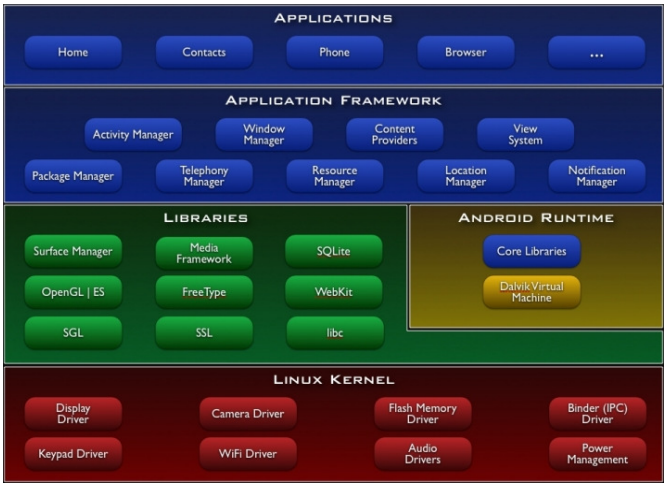

Fig 1. Camadas do Sistema Operacional Android. [8]

3.1 Linux kernel

O Android usa o kernel do Linux com alguns patches, que adicionam algumas funcionalidades através de módulos do kernel. Veremos no item 4 estas funcionalidades.

3.2 Bibliotecas e serviços.

As bibliotecas básicas do sistema como a Bionic, a OpenGL/ES para trabalhar com gráficos, e a SQLite para trabalhar com banco de dados. Aqui também estão os serviços providos para as camadas superiores, incluindo a máquina virtual Java (Dalvik). A maior parte destas bibliotecas e serviços estão desenvolvidos em C e C++.

3.3 Framework

Esta camada é desenvolvida quase toda em Java, e faz a interface com as aplicações Android. Ela provê um conjunto de bibliotecas para acessar os diversos recursos do dispositivo como interface gráfica, telefonia, localizador (GPS), banco de dados persistente, armazenamento no cartão SD, etc.

3.4 Aplicações

A camada mais alta da arquitetura Android. É onde se encontram as aplicações nativas (internet, sms, calendário) e os aplicativos instalados pelo usuário. E é um dos grandes segredos do sucesso da plataforma, já que possui mais de 250.000 aplicações no Android Market, e continua crescendo cada dia que passa.

4 O KERNEL

O Android usa uma versão modificada do kernel do Linux. Dentre as principais modificações, temos:

4.1 Binder

Em todo sistema operacional com suporte à memória virtual, os processos rodam em diferentes regiões de memória. Isso significa que nenhum processo tem acesso à região de memória de outro processo. E por isso precisamos de um mecanismo de comunicação entre processos. O Android usa o Binder para a comunicação entre processos. Ele implementa um módulo no kernel em "*drivers/misc/binder.c*" para esta tarefa. Toda comunicação entre processos no Android passa pelo Binder. Para o desenvolvedor de aplicações Android, o processo é transparente, já que é abstraído pelas bibliotecas do sistema.

4.2 Ashmem

Um novo mecanismo de compartilhamento de memória, onde dois processos podem se comunicar através desta região compartilhada de memória. É mais leve e simples de usar, e tem melhor suporte a dispositiva com pouca memória, já que tem a capacidade de descartar regiões de memória compartilhada de maneira segura em caso de pouca memória disponível. Sua implementação encontra-se em "*mm/ashmem.c*".

4.3 Logger

O Android possui um sistema global de logs, implementado através de um módulo do kernel. Ele cria quatro arquivos de dispositivo em "/dev/log", cada um representando um buffer diferente:

```
# ls -l /dev/log
crw-rw--w- root   log    10,  54 1970-01-01 00:00 system
crw-rw--w- root   log    10,  55 1970-01-01 00:00 radio
crw-rw--w- root   log    10,  56 1970-01-01 00:00 events
crw-rw--w- root   log    10,  57 1970-01-01 00:00 main
```

Para as aplicações acessarem o sistema de log, basta abrir e ler ou escrever num destes arquivos de dispositivo. A implementação deste módulo no kernel encontra-se em "*drivers/misc/logger.c*".

4.4 Wakelocks

Se um dispositivo Android ficar um tempo sem ser usado, entrará em modo de baixo consumo para garantir economia de bateria. O módulo de wakelock permite que as aplicações desabilitem o mecanismo de baixo consumo. Por exemplo, se você precisar executar um processo em background que não pode ser interrompido para entrar em modo de baixo consumo, este módulo possibilita a desativação temporária deste recurso até que seu processo finalize a execução. Sua implementação encontra-se em "*kernel/power/wakelock.c*".

4.5 oom handling

Implementado em *"drivers/misc/lowmemorykiller.c"*, controla o uso de memória do sistema e mata processos se verificar que a memória disponível esta abaixo de um valor mínimo aceitável.

4.6 Timed GPIO

Possibilita acionar saídas de I/O de forma temporizada. Está implementado em "*drives/misc/timed_gpio.c*".

5 ANDROID X IOS

Atualmente, os smartphones que mais estão se destacando no mercado são os equipados com sistema operacional Android e os iPhones. Uma comparação entre esses dois

sistemas operacionais foi detalhada na Tabela 1, tomando como base o iOS 4.2 e o Android 2.3 (Gingerbread).

TABELA 1
Comparação iOS 4.2 e Android 2.3.

	iOS 4.2	Android 2.3 (Gingerbread)
Dispositivos		
Tethering	Sim	Sim
Flash	Não	Sim
Multitarefa	Sim	Sim
Funciona como Hot-spot	Não	Sim, até 8 aparelhos por Wi-Fi
Copiar e colar	Sim	Sim
Chat por vídeo	Sim, nativo	Disponível via apps
VoIP	Disponível via apps	Suporte nativo a contas SIP
eBooks	iBooks e livros digitais	Disponível via apps
Música	Sincronia automática de listas com iTunes. Streaming por apps	Player nativo, mas sincronia automática de listas com desktop válido somente pelo Winamp. Streaming via apps
Loja de música	iTunes Store (indisponível no Brasil)	Não
E-mail	Permite inúmeras contas e traz caixa de entrada unificada	Caixa de entrada unificada varia de acordo com o fabricante. Disponível via apps
Gerenciamento de apps abertas e consumo de memória	Disponível via apps	Gerenciamento nativo
Busca por voz	Não	Sim, nativo
Software de Navegação	Não	Sim, Google Navigate

6 Mercado Android

Recentes pesquisas mostram que a plataforma Android está em 48% dos modelos de celular smartphones do mundo. Grandes empresas como HTC, Samsung, Motorola e Sony Ericsson são as grandes responsáveis pelo grande volume de vendas de "celulares inteligentes" equipados com sistema operacional Android.

Segundo nota divulgada pelo chefe da divisão do Android no Google, Andy Rubin, em junho de 2011, são mais de 500 mil ativações diárias de dispositivos móveis equipados com esse sistema no mundo, incluindo celulares e tablets. [6] Fato que colabora para essa grande ascensão no mercado é o número de aplicativos gratuitos disponíveis para essa plataforma que superou em abril de 2011 a quantidade de aplicativos do concorrente iOS da Apple.

7 Estudo de Caso

Visando experimentar a utilização dos diversos recursos estudados sobre a plataforma Android, foi desenvolvida uma aplicação que consiste em um Gestor Financeiro Pessoal, que apresenta as seguintes funcionalidades:

7.1 Cadastro de contas

No cadastro o usuário é capaz de preencher a descrição da conta, o valor, o vencimento, quando deseja ser notificado sobre o vencimento da conta e também tem a opção de salvar o número do boleto da conta.

7.2 Salvar o número do boleto

Para salvar o número do boleto da conta, a aplicação utiliza o Barcode Scanner. Nela o usuário tira uma foto do boleto e tem como retorno o número do código de barras do boleto que é salvo no cadastro. Importante ressaltar que nem todos os boletos podem ser lidos, como por exemplo, boletos bancários, por ser uma limitação da API zXing. Este é um recurso opcional no cadastro de contas da aplicação.

7.3 Visualizar contas

Com essa funcionalidade o usuário é levado a uma tela que corresponde às contas cadastradas no mês atual. Nela o usuário pode visualizar todas as contas do mês atual, ou seja, as contas que já foram pagas, as contas em atraso e as que ainda estão para vencer. O usuário pode filtrar o que deseja visualizar nessa tela e pode escolher se deseja ver todas as contas, apenas as contas a pagar ou as contas já pagas. São exibidas a descrição da conta, o valor e a data de vencimento.

7.4 Editar conta

Na tela de visualizar contas ao clicar em cima de alguma conta o usuário pode escolher entre as opções: alterar conta, remover conta ou cancelar a operação de edição. Caso escolha alterar a conta, é exibida à tela de edição para alterar os campos de descrição, valor, data de vencimento, horário a ser notificado sobre o vencimento e número do boleto ou marcá-la como paga. Se escolher remover a conta, esta será deletada do banco de dados da aplicação.

7.5 Geração de gráficos

O usuário pode gerar um gráfico de barras das contas de todos os meses do ano atual, ou seja, um gráfico que permite visualizar a evolução de suas contas durante o ano.

É possível também ao usuário gerar um gráfico de pizza que contêm as contas do mês já pagas e as contas do mês que ainda não foram pagas, a fim de fornecer ao usuário a possibilidade de comparação do que ele já pagou e o que ainda terá que ser pago.

7.6 Notificação sobre conta

Uma vez que o usuário cadastrou o horário que deseja receber a notificação da conta, ao ser alertado terá as seguintes opções: Lembrar mais tarde, marcar como pago, procurar bancos mais próximos, remarcar uma data para ser notificado, enviar notificação por SMS ou por e-mail.

Caso a conta seja marcada como pago, o usuário não será mais notificado sobre esta conta.

7.7 Localização de agências bancárias mais próximas

A aplicação utiliza o GPS do dispositivo e captura a localização do usuário para juntamente com a API Google

Maps, procurar nas proximidades por agências bancárias e fornecer ao usuário, através do mapa, os locais mais próximos para efetuar o pagamento da conta.

7.8 Envio de SMS ou E-mail

É possível, ao ser notificado, que o usuário envie a notificação por SMS ou por e-mail, a notificação contêm a conta cadastrada (descrição, valor, data de vencimento e número do boleto se existir).

7.9 Gerar notificação de contas

O usuário ao clicar em "Gerar notificação de contas", pode selecionar quantas contas não pagas desejar. Depois são exibidos ao usuário todos os contatos da agenda do telefone, a fim de selecionar quais contatos receberão essa notificação. Após isso, é feita a escolha da forma de envio, que pode ser através de e-mail ou SMS. Em ambos os casos é criada uma mensagem que contêm todas as contas selecionadas com descrição, valor e data de vencimento, bem como, o valor total dessas contas. São exibidos também os nomes das pessoas que foram selecionadas e o quanto cada uma terá que pagar, ou seja, o total das contas dividido pela quantidade de pessoas. Esta divisão e notificação de contas é útil, por exemplo, em repúblicas.

8 Recursos Utilizados

Diversos recursos do Sistema Operacional Android foram utilizados para o desenvolvimento da aplicação. A seguir serão listadas suas principais características e a forma como foram utilizados.

8.1 AndroidManifest

Em toda aplicação deve existir o arquivo AndroidManifest.xml (exatamente com esse nome) no diretório raiz. O AndroidManifest fornece informações básicas sobre a aplicação para o sistema operacional, informações que o sistema precisa ter antes de rodar o código da aplicação. Além disso, o manifest também: [3]

- Nomeia o pacote Java da aplicação
- Descreve os componentes da aplicação, tais como activity, services, broadcast receivers entre outros.
- Declara as permissões que a aplicação necessita.
- Declara o level mínimo da API Android que a aplicação requer.

A Fig. 2 mostra a estrutura do arquivo AndroidManifest.xml:

```
<?xml version="1.0" encoding="utf-8"?>
<manifest>
    <uses-permission />
    <permission />
    <permission-tree />
    <permission-group />
    <instrumentation />
    <uses-sdk />
    <uses-configuration />
    <uses-feature />
    <supports-screens />
    <compatible-screens />
    <supports-gl-texture />
    <application>
        <activity>
            <intent-filter>
                <action />
                <category />
                <data />
            </intent-filter>
            <meta-data />
        </activity>
        <activity-alias>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </activity-alias>
        <service>
            <intent-filter> . . . </intent-filter>
            <meta-data/>
        </service>
        <receiver>
            <intent-filter> . . . </intent-filter>
            <meta-data />
        </receiver>
        <provider>
            <grant-uri-permission />
            <meta-data />
        </provider>
        <uses-library />
    </application>
</manifest>
```

Fig. 2. Estrutura do arquivo AndroidManifest.xml [5].

8.2 Activity

A activity geralmente representa uma tela da aplicação, e tem por objetivo tratar todos os eventos que são gerados na interação usuário-interface.

Toda activity deve, obrigatoriamente, implementar o método onCreate(bundle) que é responsável por inicializar todos os componentes que são utilizados, além de chamar o método setContentView(view) que é responsável por desenhar a interface que será apresentada ao usuário.

Toda activity possui um ciclo de vida, que nada mais é do que os estados que ela pode vir a ter durante a sua execução, que podem ser: executando, temporariamente interrompida, em segundo plano ou destruída.

A Fig. 3 descreve o ciclo de vida completo de uma activity.

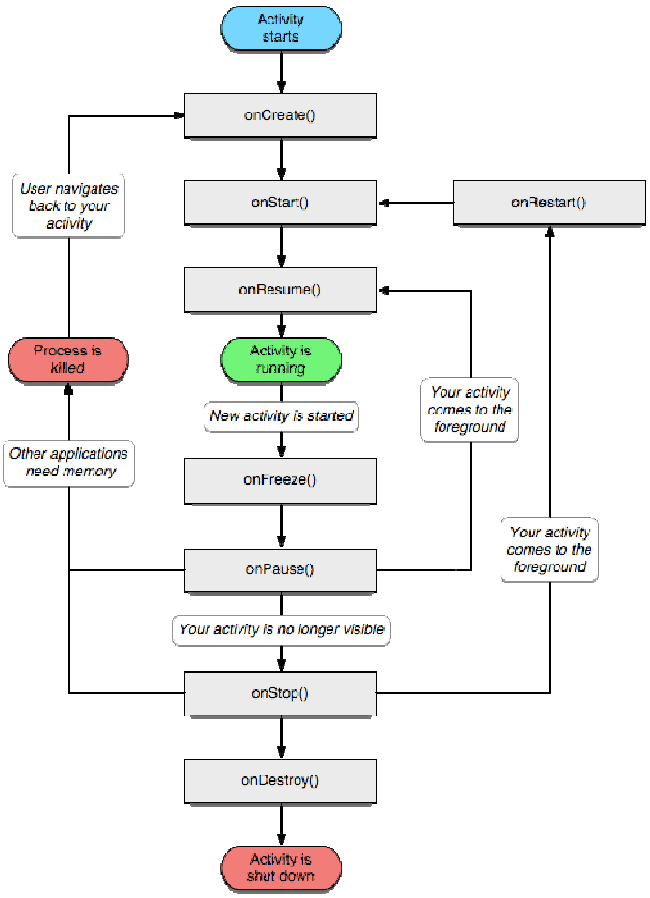

Fig. 3. Ciclo de vida de uma activity [3].

Na prática os métodos que se encontram dentro da activity são usados para controlar os diversos estados da aplicação. São eles:

onCreate(Bundle): Esse método é chamado uma única vez, nele deve-se criar a view (interface) através do comando setContentView(view).

onStart(): Esse método é chamado quando a activity está ficando visível ao usuário a já foi definida uma view para a activity.

onRestart(): É chamado quando a activity foi temporariamente parada e está sendo reiniciada.

onResume(): Esse método é chamado quando a activity está no topo da pilha, ou seja, está sendo executada com activity principal e está pronta para interagir com o usuário.

onPause(): Caso ocorra algum evento como por exemplo o celular entrar em modo de espera entre alguns outros eventos, a activity que está no topo da pilha pode ser interrompida, esse método é chamado para que seja possível salvar o estado da aplicação, para que estes dados sejam recuperados posteriormente quando a activity for reiniciada.

onStop(): Este método é chamado quando a activity está sendo encerrada e não está mais disponível para o usuário.

onDestroy(): É chamado remover a activity da pilha e liberar todos os recursos que eram utilizadas por ela.

Diversas "activities" foram utilizadas no projeto para gerar as interfaces, a Fig. 4 demonstra alguma dessas interfaces.

Fig. 4. Utilização de Activities.

8.3 Intent

A intent é um componente muito importante na arquitetura do Android, ela representa uma mensagem da aplicação para o sistema operacional solicitando que algo seja realizado, é responsável também pela facilidade encontrada no Android de integrar diferentes aplicações.

A intent pode ser definida como a intenção da aplicação de realizar uma determinada tarefa, na prática essa intenção é enviada pela aplicação para o sistema operacional em forma de uma mensagem, e este tomará as decisões necessárias de acordo com a mensagem recebida.

Uma intent pode ser utilizada para:

- Enviar uma mensagem ao SO.
- Abrir uma nova tela na aplicação.
- Solicitar ao SO para realizar uma ligação telefônica.
- Abrir o browser com um determinado endereço.
- Executar algum processamento pesado utilizando as classes broadcast receiver e service.
- Solicitar que outra aplicação realize algum processo.
- Abrir o Android Market e instalar alguma aplicação.
- Entre outras...

No projeto desenvolvido, a intent é dos componentes mais utilizados. Foi essencial em:

- Comunicação entre Activities e services.
- Solicitar ao Google Maps que exiba as agências bancárias mais próximas do local em que o usuário se encontra.
- Envio de SMS e e-mail.
- Listar os contatos gravados no celular.
- Instalar a aplicação Barcode Scanner, responsável pela leitura de códigos de barras, diretamente do Android Market.
- Solicitar ao Barcode Scanner para que faça a leitura de um código de barra.

A Fig. 5 mostra o Google Maps indicando a localização de agências bancárias.

Fig. 5. Localizando agências bancárias no Google Maps.

8.4 *IntentFilter*

A classe IntentFilter é usada basicamente para configurar qual intent será interceptada com base em seu conteúdo. Na arquitetura Android, diversas intents "circulam" com alguma mensagem, a ideia do IntentFilter é criar um "filtro" que seja capaz de reagir toda vez que uma intent com a mensagem desejada for "capturada".

No Gerenciador Financeiro foi criado uma IntentFilter que intercepta a intent que é enviada pelo Android toda vez que se passa um minuto. Esse "filtro" é utilizado para registrar um BroadcastReceiver para que este execute um método previamente especificado.

8.5 *Service*

A classe Service é utilizada quando é necessário executar um serviço em segundo plano, geralmente vinculado a um processo que deve ser executado por tempo indeterminado e requer alto consumo de recursos, memória e CPU.

Um service não possui interface gráfica e não interage com o usuário. Seu objetivo é simplesmente executar uma tarefa pesada por um tempo indeterminado.

Uma thread possui o mesmo objetivo de um service, então é comum surgir dúvidas do tipo "Qual a diferença entre elas?" e "Quando usar cada uma delas?". De forma geral, o Android conhece a classe Service e pode gerenciar seu ciclo de vida junto com os demais processos do Sistema Operacional. Em outras palavras, se uma thread for utilizada, o Android pode eliminá-la, pois ela não faz parte do seu ciclo de vida conhecido. A classe Service tem um ciclo de vida controlado e tem prioridade alta sobre qualquer outro processo executado em segundo plano, ou seja, um service só será eliminado em condições críticas de memória. Por exemplo, se uma activity inicia uma thread e logo depois é encerrada, a thread corre um grande risco de ser eliminada assim que o Android matar o processo da activity.

No aplicativo Gerenciador Financeiro, verificou-se a necessidade de criar uma thread que acessasse o banco de dados a cada minuto para verificar a existência de uma conta a ser notificada. Essa thread deveria ter uma prioridade alta para não ser eliminada pelo Android, visto que a eliminação dessa thread causaria a não notificação de uma conta, o que para o usuário seria algo muito ruim. Para evitar que isso ocorresse, a melhor opção encontrada foi criar um Service.

8.6 *Notification*

A classe Notification é utilizada para exibir uma notificação ao usuário, que é uma mensagem especial que aparece na barra de status do celular para chamar sua atenção.

Ao receber essa notificação o usuário pode decidir visualizar seu conteúdo ou simplesmente ignorar essa mensagem. Para visualizar o conteúdo, a notificação pode disparar uma intent para iniciar diversos tipos de processamentos.

A necessidade de utilizar uma notification é que uma aplicação rodando em segundo plano nunca deve exibir um alerta para o usuário sem a sua permissão. Para evitar isso, é exibida uma mensagem na barra de status do celular para chamar a atenção do usuário. Essa notificação pode inclusive fazer o celular vibrar ou ascender luzes.

No Gerenciador Financeiro a classe Notification foi utilizada para chamar a atenção do usuário quando for detectada uma conta que está prestes a vencer. Se o usuário desejar ver seu conteúdo, uma intent é lançada ao sistema operacional solicitando que uma activity mostre os detalhes dessa conta. A Fig. 6 mostra uma notificação de conta a ser paga.

Fig. 6. Notificação de conta a ser paga.

8.7 Broadcast Receiver

BroadcastReceiver é uma classe muito importante na arquitetura Android, utilizada para que aplicações possam reagir a determinados eventos gerados por uma intent, que nada mais é do que mensagens enviadas ao Sistema Operacional.

O BroadcastReceiver sempre é executado em segundo plano durante pouco tempo e sem utilizar uma interface gráfica. Seu objetivo é receber uma mensagem (intent) e processá-la sem que o usuário perceba. Isso é um importante passo para integrar aplicações, uma vez que elas podem trocar mensagens em segundo plano sem atrapalhar o usuário. [2]

O ciclo de vida de um BroadcastReceiver é bem simples. Um objeto de BroadcastReceiver só é válido durante a chamada de seu método onReceive(Context, Intent), ou seja, assim que o código retornar dessa função, o Sistema Operacional considera que o objeto já foi usado e o desativa. [7]

Um broadcastReceiver foi implementado no Gerenciador Financeiro com a finalidade de receber uma intent que é enviada pelo Android a todo minuto e executar um método que verifica se existe uma conta a ser notificada nesse momento.

8.8 SQLite

O sistema operacional Android suporta o SQL Lite, um robusto e poderoso banco de dados.

É permitida a criação de um ou mais bancos de dados em cada aplicação, e o banco de dados é visível somente para a aplicação que o criou. Eles ficam armazenados internamente na pasta: /data/data/nome-pacote/databases/

O Android permite a criação do banco de dados dentro da própria aplicação. Para isso é necessário implementar o script de criação do banco de dados e através método onCreate, fornecido pelo Android, o script é executado e o banco de dados criado.

8.8.1 Content Values

Para trabalhar com banco de dados o Android oferece a classe ContentValues e um objeto dessa classe permite definir chave/valor para inserção e atualização no banco de dados. A chave é a coluna e valor é o valor para esta coluna.

8.8.2 Cursor

É responsável por fornecer acesso de leitura e escrita para um conjunto de resultados retornados por uma consulta ao banco de dados.

8.8.3 ASyncTask

A classe AsyncTask fornece um mecanismo efetivo para trabalhar com operações que demandam tempo e necessitam dar um retorno ao usuário. Além disso, ela é uma alternativa mais simples do que trabalhar com threads, pois sua implementação é mais fácil.

Para usar o AsyncTask é necessário criar uma classe que herde de AsyncTask e implementar alguns métodos dessa classe, bem como, definir os Generics para essa classe.

Cada Generic possui uma função específica. O primeiro define o tipo que poderá ser enviado para o método doInBackground. O segundo define o tipo que pode ser usado no progresso de alguma execução que esteja sendo realizada no método doInBackgroud. E o terceiro e último define o tipo de retorno do método doInBackground.

O mecanismo do AsyncTask oferece quatro métodos para trabalhar com ele que são os seguintes:

O método onPreExecute() – É o primeiro método executado.

O método doInBackground(Params…) – É a thread background do mecanismo. Sempre é chamado após a finalização do onPreExecute. Nesse método é que a operação que demanda tempo deve ser executada.

O método onProgressUpdate(Progress…) – É chamado pela thread principal pelo método publishProgress(Progress…), ele tem por finalidade atualizar a interface com usuário para mostrar o progresso de execução da thread background.

O método onPostExecute(Result) – É chamado após a finalização do método doInBackground, esse método encerra o ciclo do mecanismo AsyncTask.

Para invocar o mecanismo basta em algum ponto de uma activity, fazer a chamada ao método execute() da classe que herda de AsyncTask.

8.9 Persistência de Dados na Aplicação

Na aplicação desenvolvida todo o armazenamento de dados utilizou o banco de dados SQLite juntamente com ContentValues e Cursor para realizar a manipulação do banco de dados como inserção, remoção, atualização e consultas, bem como o mecanismo do AsyncTask em operações que demandam tempo para serem executadas.

8.10 ContactsContract.Contacts

Essa classe representa a tabela de contatos do telefone, ela contém um registro para cada contato cadastrado no telefone, ou seja, cada registro representa uma pessoa e os seus contatos (nome, e-mail, telefone e entre outros).

Com essa classe é possível realizar operações de manipulação com os contatos, as operações possíveis são as seguintes: inserir um novo contato, atualizar um contato, excluir e fazer consultas à tabela de contatos.

Na aplicação desenvolvida a classe ContactsContract.Contacts foi utilizada para carregar na aplicação todos os contatos armazenados no telefone, afim de selecionar quais desses contatos serão notificados sobre uma ou mais contas.

8.11 Geração de Gráficos

Para a geração dos gráficos foi utilizada a API AChartEngine na versão 0.7.0. Essa biblioteca de software gráfico foi construída especificamente para trabalhar na plataforma Android e fornece muitos recursos e tipos de gráficos.

Os gráficos que foram construídos são o de barra e o de pizza. A API trabalha igualmente para ambos os gráficos,

basta fornecer um conjunto de dados e um conjunto de configurações como (legenda, cores de fonte, cores do gráfico, entre outros) e a biblioteca se encarrega de gerar o gráfico e fornecer a interação com o mesmo, permitindo a utilização de zoom e movimentação do gráfico na tela.

Apesar de a API AChartEngine ser recente e estar no início do desenvolvimento, já apresenta grande facilidade de utilização. Ela fornece exemplos para realizar a customização e implementação de novos gráficos. Com isso, a elaboração dos gráficos necessitou de um tempo para entendimento da API e após a consolidação da maneira de operação, a implementação foi simples e o resultado final satisfatório.

8.12 API ZXing (Barcode Scanner)

Para ler e capturar o número do boleto foi utilizada a API zXing na versão 1.7 e a aplicação BarcodeScanner na versão 3.7. A API zXing que é open-source, é uma biblioteca de software que realiza processamento de imagens de código de barras 1D/2D.

O foco da API é fornecer a funcionalidade de usar câmera dos telefones celulares para realizar a decodificação de códigos de barras no aparelho, sem a necessidade de comunicação com um servidor.

A aplicação BarcodeScanner que é desenvolvida pela zXing solicita a ação via intent e o BarcodeScanner realiza o trabalho de varredura e decodificação do boleto. [4].

9 Android Market

Com o crescimento dos smartphones, a instalação de aplicativos sem a necessidade de se conectar a um computador tornou se uma necessidade não só para usuários experientes, mas também para os iniciantes facilitando o uso dos smartphones, através das lojas oficiais de aplicativos. Essas lojas concentram os aplicativos específicos de cada plataforma, possibilitando ao usuário navegar até a loja e fazer a busca e instalação de aplicativos de sua escolha diretamente do aparelho, sem a necessidade de um computador.

O Android Market é a loja oficial de aplicativos da plataforma Android, onde se concentram aplicativos exclusivos para o sistema. Embora seja chamado de "loja" o Market possui tanto aplicativos pagos como gratuitos que podem ser acessados através de um aplicativo do Market que vem instalado em qualquer aparelho que utilize Android. Para ter acesso ao Market, basta o usuário ter uma conta do Google ativa no smartphone.

Mesmo o Market sendo a loja oficial, nada impede que aplicativos sejam adquiridos de outras fontes, porém a maneira mais segura é fazer a instalação direta do Market, que possui controles de segurança sobre os aplicativos disponibilizados.

Para os desenvolvedores, o Market é a melhor opção para divulgação de seus aplicativos, pois além de ser a loja oficial, qualquer pessoa pode disponibilizar seu aplicativo para download. Para isso basta possuir uma conta no Google e pagar uma taxa única que atualmente é de $25,00.

Conforme a Fig. 7, o aplicativo Gerenciador Financeiro foi disponibilizado no Market para download na modalidade gratuita, após ter sido feito o cadastro no Google e o pagamento da taxa.

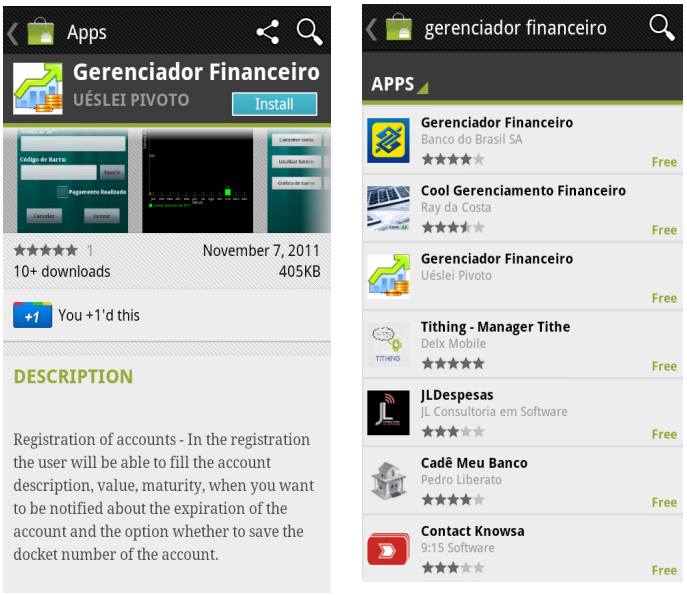

Fig. 7. Gerenciador financeiro no Android Market

10 Conclusões

Esse trabalho apresentou, de forma sucinta, uma visão geral sobre o sistema operacional Android, possuindo um foco maior no desenvolvimento de aplicações. Para isso, foi desenvolvido um aplicativo de gestão financeira que foi disponibilizado no Market Place.

A experiência de desenvolver esse aplicativo possibilitou a vivência da equipe em todo o ciclo de desenvolvimento de uma aplicação para a plataforma Android, desde a implementação e análise até a publicação. Ficou evidente que a facilidade encontrada no desenvolvimento se deve ao fato do Android possuir uma plataforma de desenvolvimento muito rica e de fácil aprendizagem, que possibilitou que a aplicação possuísse muitos recursos e funcionalidades, que poderão auxiliar a vida dos usuários.

Percebemos o quanto foi dinâmica a história e a evolução do Android e como ele tem se destacado frente aos seus concorrentes conquistando cada vez mais usuários e apoio.

Como trabalhos futuros, fica a possibilidade de adicionar recursos à aplicação para que ela receba novas funcionalidades, bem como, a criação de novos aplicativos que possam utilizar recursos da plataforma Android para contribuir com a sociedade e os usuários em geral.

Referências

[1] Lúcio Camilo Oliva Pereira, Michel Lourenço da Silva, "*Android Para Desenvolvedores*" (livro). Rio de Janeiro: Brasport, 2009.

[2] Ricardo R. Lecheta, "*Google Android: Aprenda a Criar Aplicações para Dispositivos Móveis com o Android SDK*" (livro). São Paulo: Novatec, 2010.

[3] Android Developers. public class Activity, acessado a 3 de novembro de 2011, em http://developer.android.com/reference/android/app/Activity.html

[4] code.google.com. ZXing (“Zebra Crossing”), acessado a 2 de agosto de 2011, em http://code.google.com/p/zxing/.

[5] Android Developers. The AndroidManifest.xml File, acessado a 7 de novembro de 2011, em http://developer.android.com/guide/topics/manifest/manifest-intro.html.

[6] Da Reuters (2011, 28 de junho). Sistema Android tem 500 mil ativações por dia, diz Google, acessado a 26 de setembro de 2011, em http://g1.globo.com/tecnologia/noticia/2011/06/sistema-android-tem-500-mil-ativacoes-por-dia-diz-google.html.

[7] Android Developers. public abstract class BroadcastReceiver, acessado a 1 de novembro de 2011, em http://developer.android.com/reference/android/content/BroadcastReceiver.html.

[8] Sérgio Prado (2011, 15 de agosto). Introdução ao Funcionamento Interno do Android, acessado a 20 de outubro de 2011, em http://www.sergioprado.org/2011/08/15/introducao-ao-funcionamento-interno-do-android/.