DIEGO D'LEON NUNES
DIÓGENES APARECIDO REZENDE

APLICATIVO PARA CONSULTA DE NOTAS

UNIVERSIDADE DO VALE DO SAPUCAÍ
POUSO ALEGRE – MG
2015

SUMÁRIO

1 QUADRO TEÓRICO

Neste capítulo serão descritos os principais conceitos e características das tecnologias a serem utilizadas para o desenvolvimento dos *softwares* propostos nos objetivos dessa pesquisa.

1.1 *Java*

Segundo Deitel e Deitel (2003) o java veio ao publico em 1995 pela *Sun Microsystem*. Os criadores dessa nova tecnologia liderados por James Gosling basearam-se em duas linguagens muito utilizadas no mundo, C e C++. Isso deu ao java uma base para implementar em novos sistemas como, sistemas operacionais, sistemas de comunicações, sistema de banco de dados e aplicativos para computadores pessoais.

Entre as principais características pode – se dizer que o Java é:

- Orientado a objeto.
- Seguro.
- Independente de plataforma.

Deitel e Deitel (2003) O *java* será utilizado para codificar e criar regras para proteger o banco de dados , gerenciando a infraestrutura que o próprio java fornece como, Transação, Acesso remoto, Web services, Gerenciamento de threads, Gerenciamento de conexões http[1].

Para acessar o servidor precisamos de um serviço *web* (*Web services*) que utilizara o *java* para criar as regras e serviços de compilação de dados. Segundo Deitel e Deitel (2003) existem cinco fases para que os dados cheguem em seu dispositivo que são, armazenamento em disco, o compilador cria os *bytecodes*, transfere-os para memória, verifica a sua integridade, para que não haja nenhuma violação das restrições de segurança e por ultimo o interpretador(JMV[2]) lê os *bytecodes* e os traduz para a linguagem que o computador entenda e possivelmente armazena os valores dos dados enquanto executa o programa.

Romanato (2015) afirma, que o *Java* usa a JMV , que é uma máquina virtual capaz de converter os *bytecodes* para a linguagem do sistema operacional utilizado pelo cliente, sem a ne-

[1] HTTP - HyperText Transfer Protocol.
[2] JVM - Java Virtual Machine

cessidade de compilá-lo para cada plataforma. Dessa maneira um *software* que foi desenvolvido para um sistema operacional, funcionará normalmente em qualquer outro.

1.2 *Android*

Segundo Monteiro (2012), *Android* é um sistema operacional baseado em *Linux*, de código aberto e que utiliza a linguagem de programação *Java* para o desenvolvimento de seus aplicativos. Criado especialmente para dispositivos móveis, começou a ser desenvolvido no ano de 2003 pela então empresa Android Inc, que em 2005 foi agregada ao Google. A partir de 2007 o projeto *Android* uniu-se a *Open Handset Alliance*, uma associação de empresas de *softwares*, *hardwares* e telecomunicações, que tem por finalidade desenvolver uma plataforma para dispositivos móveis que seja completa, aberta e gratuita.

Krazit (2009) publicou uma entrevista com Rubin, um dos idealizadores do *Android*, o qual afirma que o sistema pode rodar em equipamentos de diversos fabricantes, evitando assim ficar limitado a poucos dispositivos. Conforme informações do site Android (2015a), hoje em dia existe mais de um bilhão de aparelhos espalhados pelo mundo com esse sistema operacional.

De acordo com Monteiro (2012) as aplicações são executadas em uma máquina virtual Java denominada *Dalvik*. Cada aplicativo, usa uma instância dessa máquina virtual tornando-a assim mais segura. Por outro lado os *softwares* só podem acessar algum recurso do dispositivo, como uma lista de contatos, caso seja formalmente aceito pelo usuário nos termos de uso ao instalá-lo.

As configurações de uma aplicação na plataforma *Android* ficam salvas em um arquivo XML denominado AndroidManifest.xml que se localiza na raiz do projeto. Para Lecheta (2010) as informações devem estar entre tags correspondentes ao recurso. De forma, que o arquivo se inicie pela tag <manifest> e se encerre por </manifest>. Com necessidade de acessar a internet é preciso declarar a permissão da seguinte forma <uses-permission android:name="android.permission.internet"/>.

Lecheta (2010), diz que as *intents* podem ser consideradas como o coração do *Android*, uma vez que essa classe está presente em todos os lugares de uma aplicação. De acordo com K19 (2012, p.29), "são objetos responsáveis por passar informações, como se fossem mensagens, para os principais componentes da API do *Android*, como as *Activities*, *Services* e *Broad-Cast Receivers*". Monteiro (2012) diz que as *Intents* são criadas quando se tem a intenção de realizar algo como por exemplo compartilhar uma imagem, utilizando os recursos já existentes

no dispositivo. Existem dois tipos de Intents:

- *Intents* implícitas - Quando não é informada qual *Activity* deve ser chamada, ficando assim por conta do sistema operacional verificar qual a melhor opção.
- *Intents* explicitas - Quando é informada qual *Activity* deve ser chamada. Usada normalmente para chamar *activities* da mesma aplicação.

Segundo K19 (2012) uma aplicação *Android* pode ser construída com quatro tipos de componentes: *Activity*, *Services*, *Content Providers* e *Broadcast Receivers*.

As *activities* são as telas com interface gráfica, que permitem interações com os usuários. De acordo com Lacheta (2013), cada *activity* tem um ciclo de vida, uma vez que ela pode estar sendo executada, estar em segundo plano ou totalmente destruída.

Toda vez que é iniciada uma *activity*, ela vai para o topo de uma pilha denominada *activity stack*. O bom entendimento do ciclo de vida é importante, pois quando uma aplicação é interrompida é possível salvar as informações ou ao menos voltar ao estágio a qual o usuário se encontrava.

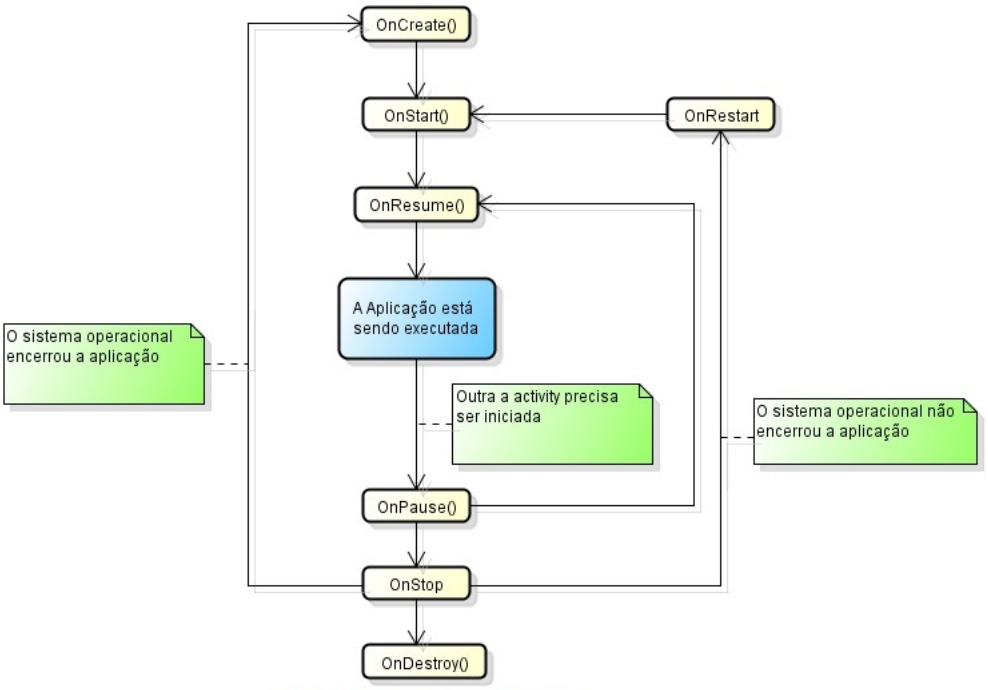

Figura 1 – Ciclo de Vida de uma *Activity*. **Fonte:**Lecheta (2010)

Na figura 1 é demostrado um exemplo de ciclo de vida de uma *activity*. Para que se possa entender melhor imagina- se o seguinte cenário: Um usuário entra no aplicativo de notas da Univás. Para que a *activity* seja criada é chamado o método *OnCreate()*, logo após é executado o método *OnStart()* e ao finalizar do ciclo anterior é chamado o *OnResume()*, só a partir de

então, a *activity* é visualizada pelo usuário. Contudo, durante a navegação o aluno recebe uma ligação. Nessa hora o sistema operacional chama o método *OnPause()* para pausar a aplicação e abrir uma outra *activity* para que o usuário possa atender a chamada telefônica. É possível nesse método salvar informação da qual o usuário está utilizando. Ao concluir o método de pausa é executado o método *OnStop()*, a partir de agora a *activity* da Univás não será mais visível para o usuário. Ao encerrar a ligação, há dois caminhos possíveis de se percorrer, o primeiro, é o caso do sistema operacional não encerrar completamente a aplicação da memória, dessa forma será chamado o método *OnRestart()* do aplicativo de notas e voltar de onde o usuário estava, porém se foi necessário encerrar completamente a aplicação, devido à falta de memória, será necessário chamar o método *OnCreate()* novamente. Por fim quando o usuário sair da aplicação é chamado o método *OnDestroy()* encerrando se assim o ciclo de vida.

No arquivo AndroidManifest.xml as *activities* devem estar entre as tags <*activity*> </*activity*> e a *activity* principal, ou seja, pela qual será iniciada a aplicação deve conter a tag <*intente-filter*> além de <*action android:name="android.intent.action.MAIN"*/> indicando que essa atividade deverá ser chamada ao iniciar a aplicação e <*category android:name="android.intent.category.LAUNCHER"*/> que implica que esse APP ficará disponível junto aos outros aplicativos no dispositivo.

A *Activity* a ser utilizada para iniciar a aplicação é uma *Navigation Drawer*. Segundo o site Android (2015b) ela exibe do lado esquerdo as principais funções do *software*, semelhante a um menu. Fica normalmente escondida aparecendo apenas quando se clica no canto superior esquerdo.

Segundo Lecheta (2010) a classe *Services* existe com intuito em executar processos que levarão um tempo indeterminado para serem executados e que normalmente consomem um alto nível de memória e processamento. Esses processos são executadas em segundo plano enquanto o cliente realiza outra tarefa. Assim um usuário pode navegar na internet enquanto é feito um *download*. O serviço é geralmente iniciado pelo *Broadcast Receiver* e quem o gerencia é o sistema operacional que só o finalizará ao concluir a tarefa, salvo quando o espaço de memória é muito baixo.

Para Lecheta (2010), a função da classe *Content Provider* é prover conteúdos de forma pública para todas as aplicações, dessa forma usando-se essa classe é possível as aplicações consultar, salvar, deletar e alterar informações no *smartphone*. Assim afirma Lecheta (2010, p.413) "O *Android* tem uma série de provedores de conteúdo nativos, como, por exemplo, consultar contatos da agenda, visualizar os arquivos, imagens e vídeos disponíveis no celular". Portanto, um contato pode ser salvo por um aplicativo e alterado por outro.

Para Lecheta (2010), a classe *Broadcast Receiver* é muito importante para a plataforma *Android*, uma vez que ela é responsável por agir em determinados eventos de uma *intent*.

Essa classe sempre é executada em segundo plano, de forma que o usuário não veja o que está sendo executado. Portanto, quando uma pessoa está utilizando uma aplicação e recebe uma mensagem de SMS, o *Broadcast Receiver* capta essa informação e a processa sem a necessidade do cliente ter que parar de realizar suas tarefas.

A configuração de um *Broadcast Receiver* é escrita pela tag <*receiver*> junto com a tag <*intente-filter*> que conterá o que deve ser feito dentro do AndroidManifest.xml.

Em uma aplicação, um elemento fundamental é a interface gráfica, que deverá ser organizada, simples e elegante. Conforme Monteiro (2012) esses são os principais *Layouts* do sistema operacional Android:

- *LinearLayout* - Permite posicionar os elementos em forma linear, dessa forma quando o dispositivo estiver em forma vertical os itens ficaram um abaixo do outro e quando estiver na posição horizontal eles ficaram um ao lado do outro.

- *RelativeLayout* - Permite posicionar elementos de forma relativa, ou seja um item com relação a outro.

- *TableLayout* - Permite criar *layouts* em formato de tabelas. O elemento *TableRow* representa uma linha da tabela e seus filhos as células. Dessa maneira, caso um *TableRow* possua dois itens significa que essa linha tem duas colunas.

- *DatePicker* - *Widget* desenvolvido para a seleção de datas que podem ser usadas diretamente no *layout* ou através de caixas de diálogo.

- *Spinner* - *Widget* que permite a seleção de itens.

- *ListViews* - Permite exibir itens em uma listagem. Dessa forma, em uma lista de compras, clicando em uma venda é possível listar os itens dessa venda selecionada.

- *Menus* - Um item muito importante, pois apresenta aos usuários as opções existentes no aplicativo.

- *AlertDialog* - Apresenta informações aos usuários através de uma caixa de diálogo. Comumente utilizado para perguntar ao cliente o que deseja fazer quando seleciona algum elemento.

- *ProgressDialog* e *ProgressBar* - Utilizado quando uma aplicação necessita de um recurso que será demorado, como por exemplo, fazer um *download*, pode ser feito uma animação informando ao usuário o progresso da operação.

Para uma maior interação, as aplicações normalmente utilizam API's de terceiros, como o Google *Maps*, quando necessita encontrar alguma localização. Para Monteiro (2012) essa comunicação pode utilizar o REST[3], que envia requisições através da URL. Ao receber informações pedidas a um outro serviço, ela pode estar no padrão XML ou JSON. O REST será detalhado mais adiante.

Outra ferramenta importante e muito utilizada do Android é a Notificação. Segundo Phillips e Hardy (2013) quando uma aplicação está sendo executada em segundo plano e necessita comunicar-se com o usuário, o aplicativo cria uma notificação. Normalmente as notificações aparecem na barra superior, o qual pode ser acessado arrastando para baixo a partir da parte superior da tela. Assim que o usuário clica na notificação ela cria uma *activity* abrindo a aplicação em questão.

Com a ideia de desenvolver um aplicativo para dispositivos móveis, a plataforma Android foi escolhida devido ao seu destaque no mercado e pela facilidade que apresenta aos usuários.

1.3 Android Studio

Umas das ferramentas mais utilizadas para o desenvolvimento em Android é o *Eclipse IDE*, contudo a Google criou um *software* especialmente para esse ambiente, chamado *Android Studio*. Segundo Gusmão (2014), *Android Studio* é uma IDE baseado no ItelliJ *Idea* e foi apresentado na Conferência para desenvolvedores I/O de 2013.

De acordo com Hohensee (2013) o *Android Studio* tem um sistema de construção baseado em *Gradle*, que permite aplicar diferentes configurações no código quando há necessidade de criar mais de uma versão, como por exemplo, um *software* que terá uma versão gratuita e outra paga, melhorando a reutilização do código. Com o *Gradle* também é possível fazer os *downloads* de todas as dependências de uma forma automática sem a necessidade de importar bibliotecas.

[3] REST - *Representational State Transfer*.

Hohensee (2013) afirma que o *Android Studio* é um editor de código poderoso, pois tem como característica a edição inteligente, pois ao digitar já completa as palavras reservadas do sistema operacional e fornece uma organização do código mais legível.

Segundo Android (2015c) a IDE tem suporte para a edição de interface, o que possibilita ao desenvolvedor arrastar os componentes que deseja. Ao testar o aplicativo permite o monitoramento do consumo de memória e de processador por parte do utilitário.

Gusmão (2014) diz que a plataforma tem uma ótima integração com o *GitHub* e está disponível para *Windows*, *Mac* e *Linux*. Além disso os programadores terão disponíveis uma versão estável e mais três versões que estarão em teste chamadas de *Beta*, *Dev* e *Canary*.

Devido ao *Android Studio* ser uma ferramenta de fácil usabilidade e a IDE oficial para o desenvolvimento Android, esta foi escolhida como ambiente de construção do aplicativo.

1.4 *Web Services*

Nos tempos atuais, com o grande fluxo de informação que percorre pelas redes da *internet* é necessário um nível muito alto de integração entre as diversas plataformas, tecnologias e sistemas. Como uma provável solução para esse ponto, já existem as tecnologias de sistemas distribuídos. Porém essas tecnologias sofrem demasiadamente com o alto acoplamento de seus componentes e também com a grande dependência de uma plataforma para que possam funcionar. Com intuito de solucionar a estes problemas e proporcionar alta transparência entre as várias plataformas, foram criados as tecnologias *web services*.

De acordo com Erl (2015):

> No ano de 2000, a W3C (*World Wide Web Consortium*) aceitou a submissão do *Simple Object Access Protocol* (SOAP). Este formato de mensagem baseado em XML estabeleceu uma estrutura de transmissão para comunicação entre aplicações (ou entre serviços) via HTTP. Sendo uma tecnologia não amarrada a fornecedor, o SOAP disponibilizou uma alternativa atrativa em relação aos protocolos proprietários tradicionais, tais como CORBA e DCOM.

Considera-se então a existência dos *web services* a partir daí. De acordo com Durães (2005), *Web Service* é um componente que tem por finalidade integrar serviços distintos. O que faz com que ele se torne melhor que seus concorrentes é a padronização do XML(*Extensible Markup Language*) para as trocas de informações. A aplicação consegue conversar com o servidor através do WSDL que é documento que contém as regras de funcionamento do *web service*.

Segundo Coulouris et al. (2013) “Um serviço *Web*(*Web service*) fornece uma interface de serviço que permite aos clientes interagirem com servidores de uma maneira mais geral do que acontece com os navegadores *Web*”. Ainda de acordo com Coulouris et al. (2013) os clientes (que podem ser desde um navegador até mesmo outro sistema) acessam serviços *Web* fazendo uso de requisições e respostas formatadas em XML e sendo transmitidos pelo uso do protocolo HTTP. O uso dessas tecnologias tende a facilitar acomunicação entre as diversas plataformas, e atende de uma forma melhor que as técnologias já existentes. Porém para que haja uma interação transparente e eficaz, entre as diversas plataformas, é necessário uma infraestrutura um pouco mais complexa para integrar todas essas tecnologias. Essa infraestrutura é composta pelas tecnologias já citadas e por outros componentes essenciais para disponibilização de serviços *web*, como mostra a figura 2.

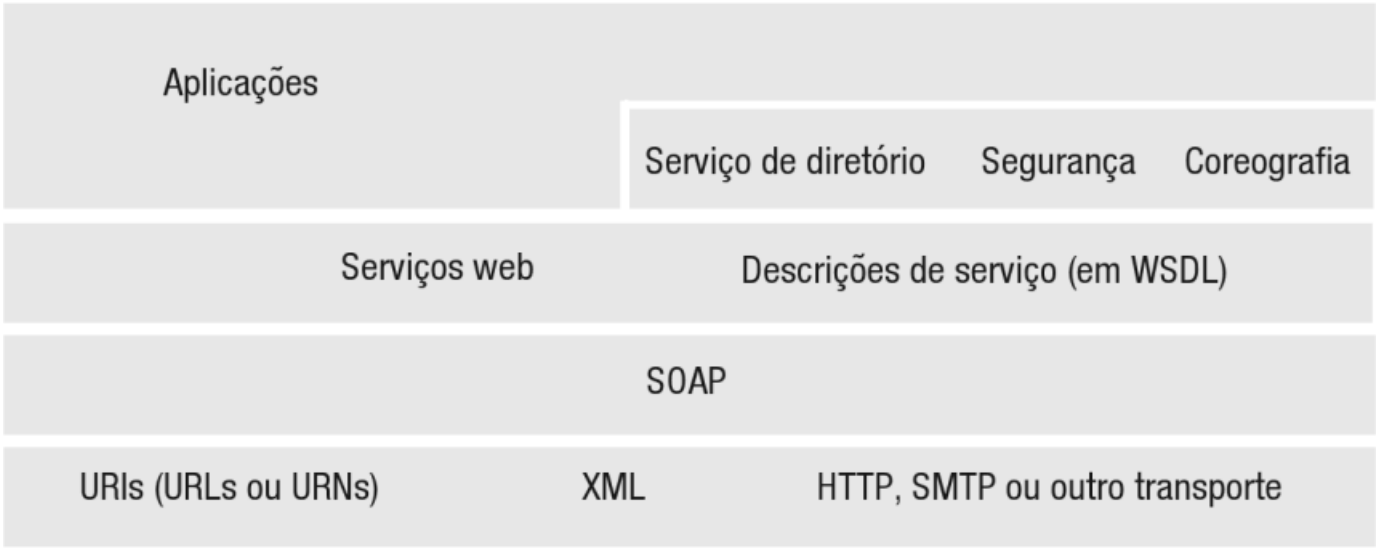

Figura 2 – Infraestrutura e componentes dos serviços *web*. **Fonte:**Coulouris et al. (2013)

Os *web services* geralmente fazem uso do protocolo SOAP, para estruturar e encapsular as mensagens trocadas. De acordo com Coulouris et al. (2013, p.381) "O protocolo SOAP(*Simple Object Access Protocol*) é projetado para permitir tanto interação cliente-servidor como assícrona pela *Internet*". Segundo Sampaio (2006, p.27) "O SOAP foi criado inicialmente, para possibilitar a invocação remota de métodos através da internet". As mensagens SOAP possuem um elemento Envelope. De acordo com Saudate (2013, p.19) "O elemento Envelope é puramente um *container* para os elementos *Header* e *Body*". Ainda de acordo com Saudate (2013) o elemento *header* transporta metadados relativos à requisição tais como autenticação, endereço de retorno da mensagem, etc. Já o elemento *body* carrega o corpo da requisição, que nada mais é do que o nome da operação e paramêtro referentes à mesma. É valido que todas requisições do trocadas usando SOAP usam o XML com formato oficial.

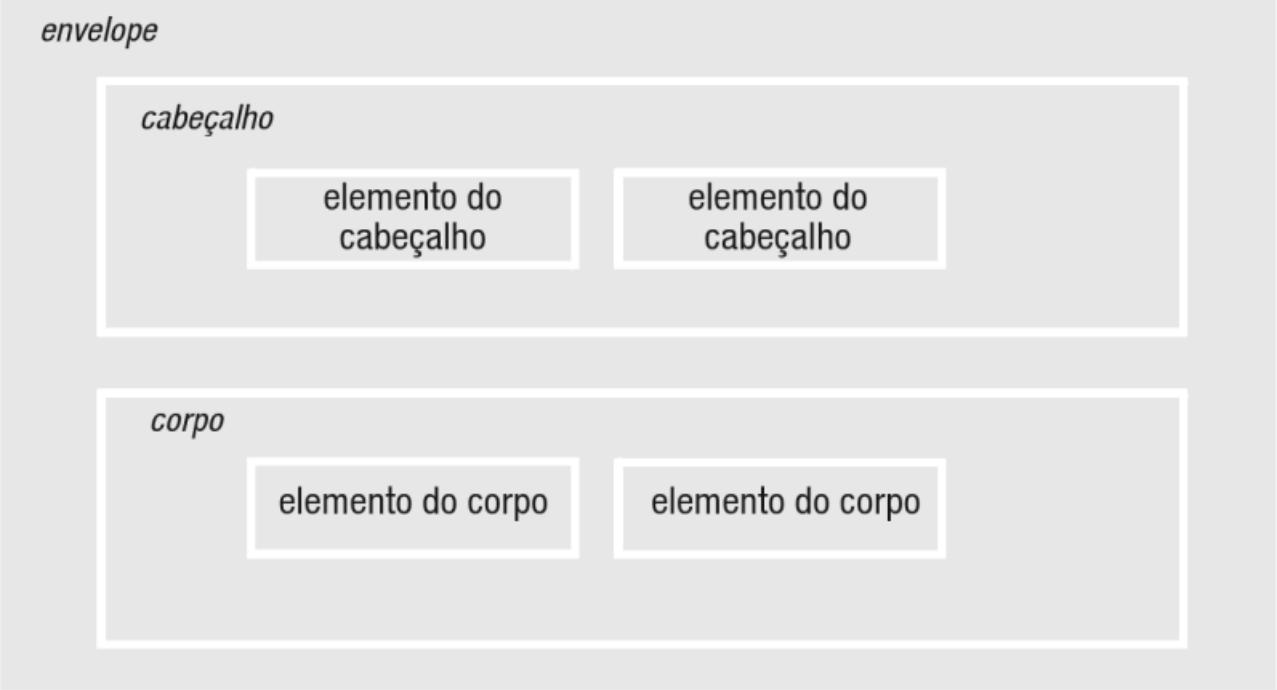

Figura 3 – Esquema de envelope SOAP. **Fonte:**Coulouris et al. (2013)

Os *web services* além de fornecerem uma padronização de comunicação entre as várias tecnologias existentes, provêem transparência na troca de informações. Isso contribui pelo fato de que as novas aplicações possam se comunicar com aplicações mais antigas ou aplicações contruídas sobre outras plataformas.

Além das tecnologias *web services* tradicionais, existe também os *web services* REST que também disponibilizam serviços, porém não necessitam de encapsulamento de suas mensagens assim como os *web Services* SOAP. Este fato influencia diretamente na performance da aplicação como um todo, haja vista que não sendo necessário o encapsulamento da informação requisitada ao *web service*, somente é necessário o processamento e tráfego da informação que realmente importa. As caracteristícas do padrão REST serão abordadas a seguir.

1.4.1 REST

Segundo Saudate (2012), REST é a sigla de *Representational State Transfer*, desenvolvido por Roy Fielding na defesa de sua tese de doutorado. Segundo o próprio Fielding (2000) REST é um estilo que deriva do vários estilos arquitectónicos baseados em rede e que combinado com algumas restrições, fornecem uma interface simples e uniforme para fornecimento de serviços[4].

[4] Tradução e resumo de informações de responsabilidade dos autores da pesquisa.

Rubbo (2015), afirma que os dados e funcionalidade de um sistema são considerados recursos e podem ser acessados através das URI's *(Universal Resource Identifier)*, facilitando dessa forma a comunicação do servidor com o cliente. Um serviço baseado na arquitetura REST(ou RestFull) basea-se fortemente em recursos. Para exemplificar o que seria um recurso em REST têm-se o seguinte cenário: considera-se uma URL tal como http://www.univas.edu.br/alunos/ onde pretende-se fazer a gerência dos dados de um ou de um conjunto de alunos. A recuperação de dados, bem como sua edição e/ou deleção fazendo uso pleno dos métodos HTTP, através dessa URL pode ser considerado como um recurso.

Saudate (2012), explica ainda que os métodos do HTTP podem fazer modificações nos recursos, da seguinte forma:

- GET – Para recuperar algum dado.
- POST – Para criar algum dado.
- PUT – Para alterar algum dado.
- DELETE – Para excluir algum dado.

Como o próprio Fielding (2000) também foi um dos criadores de um dos protocolos mais usados na web, o HTTP, pode-se dizer que o REST foi concebido para rodar sobre esse protocolo com a adição de mais algumas caracteristicas que segundo Saudate (2013) foram responsáveis pelo sucesso da web. Essas caracteristicas são:

- URLs bem definidas para recursos;
- Utilização dos métodos HTTP de acordo com seus propósitos;
- Utilização de *media types* efetiva;
- Utilização de *headers* HTTP de maneira efetiva;
- Utilização de códigos de status HTTP;
- Utilização de Hipermídia como motor de estado da aplicação.

Segundo Godinho (2009), não há um padrão de formato para as trocas de informações, mas as que mais são utilizadas é o XML[5] e o JSON[6]. O REST é o mais indicado para aplicações em dispositivos moveis, devido sua agilidade.

[5] XML - *Extensible Markup Language*.
[6] JSON - *JavaScript Object Notation*.

1.5 *Apache Tomcat*

De acordo com Tomcat (2015) *Apache Tomcat* é uma implementação de código aberto das especificações *Java Servlet* e *JavaServer Pages*. O *Apache Tomcat* é um *Servlet Container*, que disponibiliza serviços através de requisições e respostas. Caelum (2015) afirma que ele utilizado para aplicações que necessitam apenas da parte *Web* do Java EE[7].

Segundo Tomcat (2015), o projeto desse *software* começou com a *Sun Microsystems*, que em 1999 doou a base do código para *Apache Software Foundation*, e então seria lançada a versão 3.0.

Conforme Devmedia (2015), para o desenvolvimento usando o código livre com *Tomcat* é necessária a utilização das seguintes linguagens:

- JAVA - É utilizado em toda parte lógica da aplicação.
- HTML - É utilizado na parte de interação com o usuário.
- XML - É utilizado para as configurações do software.

Desta forma, o cliente envia uma requisição através do seu navegador, o servidor por sua vez a recebe, executa o *servlet* e devolve a resposta ao usuário.

1.6 PostgreSQL

Segundo Milani (2008) O postgreSQL é um SGBD(Sistema de Gerenciamento de Banco de Dados) que tem suporte para ACID(Atomicidade, consistência, isolamento e Durabilidade), que são serviços que garantem a qualidade que um banco de dados. A seguir algumas das principais características e recursos existentes no postgreSQL, que são, Replicação, Cluster(Alta disponibilidade), Multithreads, segurança ssl e criptografia.

Segundo Milani (2008):

- Replicação: É o compartilhamento de processos e distribuição das informações em diferentes bancos de dados. Ou seja as informações que serão armazenadas no servidor serão replicadas para um servidor secundário, mantendo os dados íntegros.
- Cluster: É a interligação de dois ou mais computadores e a sincronização entre eles, assim aumentando a capacidade de demanda do banco de dados.

[7] EE - Sigla para enterprise edition

- Multithreads: É a manipulação de dados de forma que mais de uma pessoa tenha acesso a mesma informação sem ocasionar atrasos ou filas de acessos.

- Segurança SSl e criptografia: Possibilita criar conexões seguras, tanta para trafegar informações de login quando aquelas consideradas sigilosas.

Stones e Matthew (2005) diz que um dos pontos fortes do postgreSQL deriva-se de sua arquitetura onde pode ser usado em um ambiente cliente/servidor, beneficiando o usuário e o desenvolvedor.

Segundo Stones e Matthew (2005) postgreSQL é comparado com qualquer outro sgbd, contendo todas as características que encontraria em outro banco de dados, e algumas características que não encontra em outro *software* como, transações, subconsultas, chave estrangeira e regras de herança.

O PostgreSQL é um *software* de fácil utilização e multiplataforma o que leva a ser implantado em muitas empresas.

1.7 Engenharia De *Software*

De acordo com Carvalho e Chiossi (2001) a engenharia de *software* surgiu na década de 80, com intuito de melhorar o desenvolvimento de *software*, produzindo sistemas de alta qualidade com a redução do custo e do tempo.

Segundo Pressman (2011, p.39) engenharia de *software* é "o estabelecimento e o emprego de sólidos princípios da engenharia de modo a obter *software* de maneira econômica, que seja confiável e funcione de forma eficiente em máquinas reais".Lecheta (2010)

Como afirma Carvalho e Chiossi (2001) a engenharia possui modelos de processos que possibilitam ao gerente controlar o desenvolvimento e aos programadores uma base para produzir. Abaixo serão citados alguns desses paradigmas:

- Ciclo de vida clássico - Utiliza o método sequencial, em que o final de uma fase é o início da outra.

- O paradigma evolutivo - Baseia-se no desenvolvimento e implementação de um produto inicial. Esse produto passa por críticas dos usuários e vai recebendo melhorias e versões até chegar ao produto desejado.

- O paradigma espiral - Engloba as melhores características do ciclo de vida clássica e o paradigma evolutivo. Ele consiste em vários ciclos e cada ciclo representa uma fase do projeto.

De toda a engenharia de *software*, o que mais será utilizado nesse projeto é a linguagem UML, que através dos seus diagramas norteará os caminhos a serem seguidos.

1.7.1 UML

De acordo com Booch, Rumbaugh e Jacobson (2012) "A UML (*Unified Modeling Language*) é uma linguagem-padrão para a elaboração da estrutura de projetos de *software*". Na decada de 80 seguindo o surgimento e evolução das linguagens de programação orientadas a objetos, foram surgindo as linguagens de modelagens orientadas a objetos, como um modo alternativo de análise e projeto de *software* que era usada na época. De acordo com Guedes (2011, p.19):

> A UML surgiu da união de três métodos de modelagem: o método de Booch, o método OMT (*Object Modeling Technique*) de Jacobson, e o método OOSE (*Object-Oriented Software Engineering*) de Rumbaugh. Estes eram, até meados da década de 1990, os métodos de modelagem orientada a objetos mais populares entre os profissionais da área de desenvolvimento de *software*. A união desses métodos contou com o amplo apoio da *Rational Software*, que a incentivou e financiou.

Segundo Booch, Rumbaugh e Jacobson (2012, p.13) "A UML é apenas uma linguagem e, portanto, é somente uma parte de um método para desenvolvimento de *software*". Ainda de acordo com Booch, Rumbaugh e Jacobson (2012, p.13) "A UML é independente de processo, apesar de ser perfeitamente utilizada em processo orientado a casos de usos, centrado na arquitetura, interativo e incremental". A linguagem de modelagem UML além de fornecer um vocabulário próprio, também provê uma série de diagramas que tem inúmeras finalidades diferentes. Tais finalidades e suas subdivisões estão descritas na figura 1.

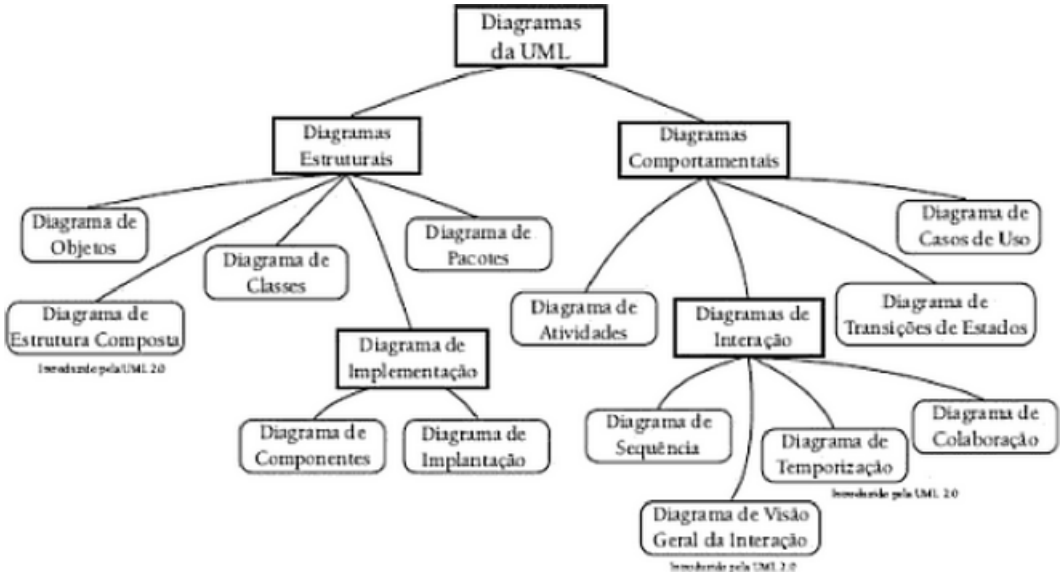

Figura 4 – Diagramas definidos pela UML. **Fonte:**Bezerra (2015)

A linguagem de modelagem UML não é um processo rígido e permite uma adequação de acordo com a situação do projeto em que é aplicada. Por permitir essa flexibilidade e prover suporte adequado para determinados casos de um projeto, é que a linguagem de modelagem UML será usada na modelagem dos *softwares* propostos nos objetivo s dessa pesquisa.

1.7.2 Processos de *Software*

Segundo Pressman (2011, p.52) um processo de *software* é "uma metodologia para as atividades, ações e tarefas necessárias para desenvolver um *software* de alta qualidade".

Para Sommerville (2003) não existe um processo ideal, pois isso dependerá de cada projeto, possibilitando cada qual implementar algum modelo já existente. Contudo Pressman (2011) afirma que uma metodologia genérica tem cinco passos:

- Comunicação - Antes de iniciar os trabalhos técnicos deve-se entender os objetivos do sistema e levantar requisitos para o bom funcionamento do *software*.
- Planejamento - Cria um plano de projeto, que conterá as tarefas a serem seguidas, riscos prováveis e recursos necessários.
- Modelagem - Esboça o sistema para que se tenha uma ideia de como ele deverá ficar e como encontrar a melhor solução para desenvolve-lo.
- Construção - É a etapa de desenvolvimento e teste.

- Emprego - O *software* pronto em sua totalidade ou parcialmente é implantado no cliente e este retorna o seu *feedback*.

Dessa forma, normalmente qualquer um dos modelos (ciclo de vida clássico, evolutivo ou espiral) utilizaram os princípios das metodologias acima citadas.

1.8 Google Cloud Messaging

Para que os alunos sejam notificados quando houver alguma mudança no portal do aluno será utilizada uma API oferecida pela *Google* denominada *Google Cloud Messaging* ou simplesmente GCM[8], um recurso que tem por objetivo notificar as aplicações Android. Segundo Leal (2014) ele permite que aplicações servidoras possam enviar pequenas mensagens de até 4 KB[9] para os aplicativos moveis, sem que esta necessite estar em execução, pois o sistema operacional envia mensagens em *broadcast*, dessa forma a aplicação pode tratar as mensagens como bem preferir.

De acordo com Leal (2014) para o bom funcionamento do recurso apresentado, são necessários os seguintes componentes:

- *Sender ID*[10] – É o identificador do projeto. Será utilizado pelo servidores da *Google* para identificar a aplicação que envia a mensagem.
- *Application ID* – É o identificador da aplicação Android. O identificador é o nome do pacote do projeto que consta no AndroidManifest.xml.
- *Registration ID* – É o identificador gerado pelo servidor GCM quando aplicação Android se conecta a ele. Deve ser enviado também a aplicação servidora.
- *Sender Auth Token* – É uma chave que é incluída no cabeçalho quando a mensagem é enviada da aplicação servidora para o GCM. Essa chave é para que a API da Google possa enviar as mensagens para o aplicativo correto.

De acordo com os componentes acima citados, quando uma aplicação servidora enviar uma mensagem para o aplicativo Android, na verdade está enviando para o servidor GCM que será encarregado de enviar a mensagem para a aplicação mobile.

[8] *Google Cloud Messaging*
[9] *Kilobytes*
[10] Identity

REFERÊNCIAS

ANDROID. : **A história do Android.** 2015. Disponível em: <https://www.android.com/history/>. Acesso em: 25 de Fevereiro de 2015.

ANDROID. : Creating a navigation drawer. 2015. Disponível em: <https://developer.android.com/training/implementing-navigation/nav-drawer.html>. Acesso em: 28 de julho de 2015.

ANDROID. : **Android Studio Overview.** 2015. Disponível em: <http://developer.android.com/tools/studio/index.html>. Acesso em: 12 de Março de 2015.

BEZERRA, E. : **Princípios De Análise E Projeto De Sistemas Com Uml**. 3ª. ed. São Paulo: Elsevier, 2015.

BOOCH, G.; RUMBAUGH, J.; JACOBSON, I. : **UML:** guia do usuário. 2ª. ed. Rio De Janeiro: CAMPUS, 2012.

CAELUM. : **Java para Desenvolvimento Web.** 2015. Disponível em: <https://www.caelum.com.br/apostila-java-web/o-que-e-java-ee/#3-4-servlet-container>. Acesso em: 15 de Fevereiro de 2015.

CARVALHO, A.; CHIOSSI, T. : **Introdução à Engenharia de Software.** Campinas: Editora da Unicamp, 2001.

COULOURIS, G. et al. : **Sistemas Distribuídos**conceitos e projeto. 5ª. ed. Porto Alegre: Bookman Editora, 2013.

DEITEL, H.; DEITEL, P. : **Java como Programar**projeto orientado a objetos com a uml e padrões de projeto. 4ª. ed. Porto Alegre: Pearson Prentice Hall, 2003.

DEVMEDIA. : **Conheça o Apache Tomcat.** 2015. Disponível em: <http://www.devmedia.com.br/conheca-o-apache-tomcat/4546>. Acesso em: 08 de Março de 2015.

DURãES, R. : **Web Services para iniciantes**. 2005. Disponível em: <http://imasters.com.br/artigo/3561/web-services/web-services-para-iniciantes/>. Acesso em: 10 de Março de 2015.

ERL, T. : **Introdução às tecnologias Web Services:** soa, soap, wsdl e uddi. 2015. Disponível em: <http://www.devmedia.com.br/introducao-as-tecnologias-web-services-soa-soap-wsdl-e-uddi-parte1/2873>. Acesso em: 26 de Abril de 2015.

FIELDING, R. T. : **Architectural Styles and the Design of Network-based Software Architectures**. Tese (Doutorado) — University of California, 2000.

GODINHO, R. : **Criando serviços REST com WCF.** 2009. Disponível em: <https://msdn.microsoft.com/pt-br/library/dd941696.aspx>. Acesso em: 01 de Março de 2015.

GUEDES, G. T. A. : **UML 2 :** uma abordagem prática. 2ª. ed. São Paulo: Novatec, 2011.

GUSMãO, G. : **Google lança versão 1.0 do IDE de código aberto Android Studio.** 2014. Disponível em: <http://info.abril.com.br/noticias/it-solutions/2014/12/google-lanca-versao-1-0-do-ide-de-codigo-aberto-android-studio.shtml>. Acesso em: 03 de Março de 2015.

HOHENSEE, B. : **Getting Started with Android Studio.** Gothenburg: [s.n.], 2013.

K19. : **Desenvolvimento mobile com Android**. 2012.

KRAZIT, T. : **Google's Rubin:** android 'a revolution'. 2009. Disponível em: <http://www.cnet.com/news/googles-rubin-android-a-revolution/>. Acesso em: 20 de Fevereiro de 2015.

LACHETA, R. R. : **Google Android: Aprenda a criar aplicações para dispositivos móveis com o Android SDK.** 3ª. ed. São Paulo: Novatec, 2013.

LEAL, N. : **Dominando o Android:** do básico ao avançado. 1ª. ed. São Paulo: Novatec, 2014.

LECHETA, R. R. : **Google Android:**aprenda a criar aplicações para dispositivos móveis com android sdk. 2ª. ed. São Paulo: Novatec, 2010.

MILANI, A. : **PostgreSQL.** São Paulo: Novatec, 2008.

MONTEIRO, J. B. : **Google Android: Crie aplicações para celulares e tablets.** São Paulo: Casa do Código, 2012.

PHILLIPS, B.; HARDY, B. : **Android Programming:**the big nerd ranch guide. Atlânta: Big Nerd Ranch, 2013.

PRESSMAN, R. : **Engenharia de software:** uma abordagem profissional. 7ª. ed. Porto Alegre: AMGH, 2011.

ROMANATO, A. : **Entenda como funciona a Java Virtual Machine (JVM).** 2015. Disponível em: <http://www.devmedia.com.br/entenda-como-funciona-a-java-virtual-machine-jvm/27624>. Acesso em: 08 de Março de 2015.

RUBBO, F. : **Construindo RESTful Web Services com JAX-RS 2.0.** 2015. Disponível em: <http://www.devmedia.com.br/construindo-restful-web-services-com-jax-rs-2-0/29468>. Acesso em: 03 de Março de 2015.

SAMPAIO, C. : **SOA e Web Services em Java**. 1ª. ed. Rio de Janeiro: Brasport, 2006.

SAUDATE, A. : **REST:**construa api's inteligentes de maneira simples. São Paulo: Casa do Código, 2012.

SAUDATE, A. : **SOA aplicado:**integrando com web serviços e além. 1ª. ed. São Paulo: Casa do Código, 2013.

SOMMERVILLE, I. : **Engenharia de Software.** 6ª. ed. São Paulo: Addison Wesley, 2003.

STONES, R.; MATTHEW, N. : **Beginning Databases with PostgreSQL:**from novice to professional. New York: Apress, 2005.

TOMCAT, A. : **The Tomcat Story.** 2015. Disponível em: <http://tomcat.apache.org/heritage.html>. Acesso em: 08 de Março de 2015.