Fundação Educacional do Município de Assis
Instituto Municipal de Ensino Superior de Assis
Campus "José Santilli Sobrinho"

MAYCO RIBEIRO DE PAIVA

ABORDAGEM DO MODELO ARQUITETURAL ORIENTADO A RECURSOS PARA INTEGRAÇÃO DE APLICAÇÕES JAVA EE E GOOGLE ANDROID

Assis
2013

MAYCO RIBEIRO DE PAIVA

ABORDAGEM DO MODELO ARQUITETURAL ORIENTADO A RECURSOS PARA INTEGRAÇÃO DE APLICAÇÕES JAVA EE E GOOGLE ANDROID

Trabalho de Conclusão de Curso apresentado ao
Instituto Municipal de Ensino Superior de Assis,
como requisito do Curso de Graduação

Orientador: Prof. Esp. Guilherme de Cleva Farto

Área de Concentração: Informática

Assis
2013

FICHA CATALOGRÁFICA

PAIVA, Mayco Ribeiro

Abordagem do modelo arquitetural orientado a recursos para integração de aplicações Java EE e *Google Android* / Mayco Ribeiro de Paiva. Fundação Educacional do Município de Assis – FEMA – Assis, 2013.

106p.

Orientador: Prof. Esp. Guilherme de Cleva Farto

Trabalho de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis – IMESA

1. Arquitetura ROA **2.** Java EE **3.** Web Services **4.** Google Android

CDD: 001.6

Biblioteca da FEMA

ABORDAGEM DO MODELO ARQUITETURAL ORIENTADO A RECURSOS PARA INTEGRAÇÃO DE APLICAÇÕES JAVA EE E GOOGLE ANDROID

MAYCO RIBEIRO DE PAIVA

Trabalho de Conclusão de Curso apresentado ao Instituto Municipal de Ensino Superior de Assis, como requisito do Curso de Graduação, analisado pela seguinte comissão examinadora:

Orientador: Prof. Esp. Guilherme de Cleva Farto

Analisador (1): Prof. Dr. Alex Sandro Romeo de Souza Poletto

Assis
2013

DEDICATÓRIA

Dedico este trabalho à minha família, meus pais, meu irmão e todas as pessoas que direta ou indiretamente contribuiu para a realização deste sonho.

AGRADECIMENTOS

Agradeço a Deus, por me dar saúde, motivação, força para vencer os obstáculos durante este período de estudos e conforto nas horas difíceis, possibilitando tornar um sonho realidade.

Ao meu orientador Guilherme de Cleva Farto, por me orientar e ajudar a enriquecer o meu trabalho com sua experiência, tornando a pesquisa melhor elaborada com suas dicas.

Aos meus amigos Daniel Herbert Hoch e Patrick Francis Gomes Rocha, por me acompanharem nesta caminhada de quatro anos de trabalhos e estudos juntos.

A minha esposa Silvana Aparecida Andrade, por me ajudar a concretizar este sonho em realidade, me confortando nas horas difíceis.

Agradeço também aos meus pais Luiza Maria Ribeiro de Paiva e Otávio Luiz de Paiva, por terem me ensinado a ser disciplinado, por estarem ao meu lado nesta caminhada e por ficarem com o meu filho Mayron Andrade de Paiva, para que eu pudesse estudar.

Ao meu irmão Rodrigo Ribeiro de Paiva e minha cunhada Erika de Castro Simongini Paiva, pela força, incentivo e conselhos que me deram durante essa caminhada de estudos.

A minha sogra Francisca, meu sogro Antônio e meu cunhado Sérgio, por ficarem com o Mayron aos sábados, para que eu pudesse estudar.

"*Um raciocínio lógico leva você de A a B.*
A imaginação te leva a qualquer lugar que você quiser".
Albert Einstein (1879 – 1955)

RESUMO

Nos dias atuais o uso da *Internet* por meio de dispositivos móveis é cada vez maior e a tendência é aumentar ainda mais. Hoje uma grande parte da população mundial fica conectada por meio de dispositivos móveis o tempo todo em redes sociais, e-mails, entre outros, fazendo uso de toda essa praticidade da vida moderna. Mas ainda a muito a ser explorado nessa área para uso corporativo pelas empresas. Diante deste cenário tão favorável este trabalho tem como intuito pesquisar e compreender as tecnologias envolvidas para fazer a integração de um aplicativo *Web* com um *Mobile*. Para isso é necessário fazer um estudo envolvendo a Arquitetura Orientada a Recursos (ROA), o conjunto de especificações existentes na plataforma *Java Enterprise Edition* (JEE), as ferramentas necessárias para utilização de *Web Services* usando a linguagem de programação Java, e por último a plataforma *Google Android*. O objetivo final deste trabalho é colocar em prática todo conteúdo teórico pesquisado, usando as tecnologias envolvidas no trabalho para desenvolver uma aplicação baseada na plataforma Java Enterprise Edition interagindo via *Web Services* como uma aplicação desenvolvida na plataforma *Google Android*.

Palavras-chave: Arquitetura Orientada a Recursos (ROA); *Java Enterprise Edition* (JEE); *Web Services*; *Google Android*

ABSTRACT

Nowadays the use of *Internet* through *Mobile* devices is increasing and the trend is increasing further. Today a large part of the world population is connected via *Mobile* devices all the time on social networks, e-mails, among others, making use of all this practicality of modern life. But still much to be explored in this area for use by corporate companies. Given this scenario as favorable this work has the intention to search and understand the technologies involved to integrate a Web application with a *Mobile*. For this it is necessary to make a study of the Resource Oriented Architecture (ROA), the set of existing specifications in the Java Platform, Enterprise Edition (JEE), the tools to use Web services using the Java programming language, and lastly Google Android platform. The ultimate goal of this work is to put into practice all the theoretical content researched, using the technologies involved in the work to develop an application based on Java Platform Enterprise Edition interacting via Web Services as an application developed on the platform Google Android.

Keywords: Resource Oriented Architecture (ROA), Java Enterprise Edition (JEE), Web Services, Google Android

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

SUMÁRIO

1 – INTRODUÇÃO

Atualmente, o volume de informação a ser compartilhada, armazenada e processada pelas empresas é cada vez maior. Diante deste cenário, torna-se necessária o desenvolvimento de soluções computacionais capazes de integrar diversas aplicações e tecnologias. Entretanto, o processo de comunicação entre *softwares* corporativos tem resistido a essa tendência positiva (PULIER; TAYLOR, 2008).

Os dispositivos portáteis vêm se destacando devido à crescente evolução da tecnologia móvel, tornando-se uma importante fonte de transmissão e recepção de informações e exigindo que os sistemas operacionais sejam cada vez mais robustos, fato este que ocasiona uma demanda muito grande para o desenvolvimento de serviços e aplicações voltadas para diversas plataformas móveis (GOMES; SOUZA; ARAÚJO, 2011).

Empresas atentas às oportunidades abertas pela conectividade que a rede móvel oferece hoje aos usuários começaram a programar modificações em seus *softwares* com a finalidade de usufruir das oportunidades que a *Web* tem a oferecer (ROVARIS, 2011).

Com a adoção dos *Web Services* juntamente com os protocolos e padrões abertos como o *Hypertext Transfer Protocol* (HTTP) e *Extensible Markup Language* (XML), tornou-se possível solucionar os inúmeros problemas de interoperabilidade entre sistemas, permitindo-se a integração de aplicações escritas em tecnologias distintas, em linguagens de programação diferentes e em sistemas operacionais diversos, proporcionando uma maior rapidez e facilidade no desenvolvimento de soluções voltadas para a tecnologia móvel (OLIVEIRA, 2012).

Com a popularização dos dispositivos portáteis, aliados a emergente infraestrutura de conectividade oferecida pela telefonia móvel, onde é possível manter-se conectado durante todo o tempo, surgiu proporcionalmente à necessidade de desenvolvimento de *softwares* e aplicações cada vez melhores para fazer uso de todo o poder disponível nestes pequenos dispositivos, tornando importante não só para o uso pessoal, mas também profissional (ABLESON; COLLINS; SEN, 2009).

Neste trabalho serão abordadas tecnologias emergentes e modernas tais como Java *Enterprise Edition* (JEE) e *frameworks* e bibliotecas relacionados como *JavaServer Faces* (JSF), Java *Persistence* API (JPA) e *PrimeFaces*, assim como o desenvolvimento de *Web Services* baseados na arquitetura orientada a recursos e a plataforma *Google Android* com a finalidade de integrar soluções corporativas *Web* e *Mobile*.

O Java EE é uma plataforma composta por um conjunto de especificações com infraestrutura capaz de facilitar o desenvolvimento de aplicações. Estas especificações faz com que o programador escreva menos código diminuindo os custos e o tempo de desenvolvimento, ou seja, o desenvolvedor foca na aplicação que esta sendo desenvolvida tirando proveito dos serviços e APIs existentes na plataforma (SAMPAIO, 2011).

Os conceitos e tecnologias acerca de *Web Services* tornam possível a integração de aplicações assim como disponibilizar tarefas simples ou complexas na forma de serviços *Web*, mesmo que as aplicações estejam em linguagens e tecnologias diferentes. Segundo Oliveira (2012, p. 14), “os *Web Services Representational State Transfer* (RESTful) surgiram como uma forma de simplificar o desenvolvimento de serviços *Web*”.

O *Google Android* é uma plataforma *open-source* criada pela *Google* e que hoje é mantida em parceria com a *Open Handset Alliance* (OHA), grupo formado por diversas empresas do ramo de computação e telefonia. A plataforma inclui um sistema operacional baseado no *Linux* e possibilita o desenvolvimento e integração de aplicações móveis de forma simplificada por meio da linguagem de programação Java (LECHETA, 2010) e (LECHETA, 2012).

1.1. OBJETIVOS

O objetivo deste trabalho é realizar a integração de uma aplicação *Web* com uma *Mobile*, utilizando-se os conceitos de arquitetura orientada a recursos e serviços *Web* desenvolvidos com tecnologias e especificações Java, no sentido de avaliar se

a integração de uma aplicação *Web* desenvolvida em Java EE e uma aplicação *Mobile* desenvolvida na plataforma *Google Android* contribui, diretamente, para a construção de soluções computacionais interoperáveis, assim como fornecer uma base sólida acerca dos conhecimentos adquiridos e fomentar pesquisas e projetos futuros na área de integração de sistemas corporativos e plataformas móveis.

Visando o sucesso desde objetivo, este por sua vez foi dividido em duas etapas. A primeira foi realizada uma pesquisa sobre a Arquitetura Orientada a Recursos (ROA), *Web Services* e a plataforma *Google Android* e na segunda etapa foi desenvolvida uma aplicação *Web* interagindo com uma aplicação *Mobile*, através de *Web Services*.

1.2. JUSTIFICATIVAS

Nos últimos anos a integração de aplicações diversas com dispositivos móveis vem crescendo de forma extraordinária e consequente proporcionando enormes facilidades e vantagens para os usuários. Cada ano que passa os dispositivos móveis vem se firmando no mercado e consequentemente as empresas de telefonia vem melhorando a qualidade e a velocidade de transmissão de dados. A facilidade e o barateamento de conexões 3G, 4G, *Wi-Max* entre outras, aliados às facilidades de se ter um dispositivo móvel, promovem a cada dia um grande número de novas contratações com as operadoras de telefonia móvel.

O *Web Service* é uma das tecnologias mais utilizadas para integrar aplicações distintas nos tempos atuais. Juntamente a esse crescimento vertiginoso onde tudo leva a transmissão de dados via *Web* por meio de dispositivos móveis, a portabilidade e integração de aplicações desenvolvidas em diversas linguagens e tecnologias é de extrema importância para profissionais da área da informática, tornando possível a comunicação entre projetos *Web* e *Mobile*.

O mercado corporativo também é outro ramo que esta em plena expansão e diversas empresas está buscando incorporar aplicações móveis ao seu cotidiano

para agilizar suas rotinas diárias e consequentemente ter suas informações atualizadas mais rapidamente e com menor custo.

1.3. MOTIVAÇÃO

Recentemente, o número de dispositivos móveis tem crescido rapidamente e a grande maioria dos usuários os utiliza basicamente para acessar redes sociais, operações *online banking*, mensagens eletrônicas entre outras aplicações.

O número de pessoas ao redor do mundo que fazem uso de *smartphones* e *tablets* com o sistema operacional *Android* é cada vez maior. O *Android* tornou-se um sistema operacional de fácil manuseio tanto para os usuários como para os programadores e, por esse e outros motivos, surgem novas aplicações para a plataforma da *Google* a cada dia.

Segundo uma pesquisa realizada nos Estados Unidos da América, Itália, França, Espanha, Austrália, Reino Unido, Alemanha e Brasil no período de agosto de 2012, revelou que a plataforma *Android* detém 60% do mercado mundial seguido do iOS da *Apple* com 24%. Analisando somente o Brasil, o *Android* ficou com 46,8%, seguido pelo *Symbian* da *Nokia* com 22%, em terceiro o *Windows Phone* com 14,9% e em quarto e último lugar aparece o iOS com 7,5%. Comparando o mesmo período de 2011, a pesquisa mostrou que o *Android* teve um crescimento de mais 30% em apenas doze meses no Brasil (BARROS, 2012).

No mundo empresarial ainda falta muito para usufruir das inúmeras vantagens tais como portabilidade, interoperabilidade e conectividade que a plataforma oferece. Na atualidade, muitas das empresas dependem de informações geradas fora das mesmas, aumentando a quantidade de informações armazenadas em formulários e criando a necessidade de se inserir os dados manualmente nos sistemas da empresa. Com aplicações corporativas *Web* interagindo com dispositivos móveis baseados em *Android* por meio do uso de *Web Services*, torna-se possível automatizar os processos de integração e persistência de dados, obtendo um ganho significativo na agilidade e qualidade dessas informações. Hoje em dia a procura por

smartphones e *tablets* cada vez mais sofisticados é grande, e isso é um caminho sem volta, a tendência é cada vez crescer mais e para suportar toda essa ambição dos usuários por novas funcionalidades, as plataformas hoje existentes, terão que sofrer atualizações constantemente.

1.4. ESTRUTURA DO TRABALHO

O trabalho esta estruturado em oito capítulos mais as referências, iniciando com uma breve introdução dos assuntos abordados no decorrer do texto. No segundo capítulo é descrito sobre a Arquitetura Orientada a Recursos (ROA), onde é abordado todo o conceito sobre este assunto. Em seguida o tema abordado é a plataforma Java *Enterprise* Edition (JEE), que é muito utilizada para desenvolvimento de aplicações *Web*. No quarto capítulo é descrito sobre uma tecnologia que surgiu no final da década de noventa para promover a integração de aplicações distintas, chamado de *Web Services*. Já no quinto capítulo é descrito sobre a tecnologia *Google Android*, que vem ganhando muito mercado nestes últimos anos. No capítulo sexto é apresentado à proposta de trabalho e no capítulo seguinte é descrito o estudo de caso que foi proposto no capítulo anterior. Por último temos a conclusão de todo o trabalho realizado, seguido das referências pesquisadas para o desenvolvimento do trabalho.

2 – ARQUITETURA ORIENTADA A RECURSOS (ROA)

A arquitetura orientada a recursos (ROA) foi desenvolvida por *Sam Ruby* e *Leonard Richardson* e publicado em seu livro *RESTful Web Services* no ano de 2007. Esta arquitetura permite converter um problema em um serviço *Web* REST, baseando-se em tecnologias sólidas da *Web* como HTTP, URI entre outros tipos de mídia. ROA apresenta um modelo de boas práticas para ser aplicadas em desenvolvimento de serviços *Web*, unificando os princípios de REST com as tecnologias da *Web*. REST e ROA juntos, reúnem vários atributos que são desejáveis em uma aplicação, tais como baixo acoplamento, melhor adaptabilidade e interoperabilidade (FRANÇA, et al., 2011).

O termo REST (*Representational State Transfer*) é um modelo arquitetural onde explora a tecnologia e os protocolos da Web, como o HTTP e o XML. Nele é definido um conjunto de regras que podem ser empregadas na elaboração de sistemas utilizando a arquitetura orientada a recursos (ROA). O REST, portanto nada mais é do que um modelo de arquitetura de software que apresenta uma definição onde pode ser empregado no desenvolvimento de sistemas chamado de sistemas *RESTful*. ROA é uma forma de converter um problema em um serviço *Web RESTful*, utilizando-se de arranjo de URIs, HTTP, XML e JSON, ou seja, usando recursos que normalmente é usado em serviços *Web*, e dessa forma facilitando no desenvolvimento de sistemas distribuídos (RICHARDSON; RUBY, 2007).

2.1. INTRODUÇÃO

A Arquitetura Orientada a Recursos surgiu devido às dificuldades que os desenvolvedores encontravam em projetar aplicações *Web* baseada em REST, visto que ele abrange uma quantidade muito ampla de possibilidades para elaboração de um projeto. Umas das dificuldades encontradas, esta no fato de que o REST não é acoplado com a *Web* e suas tecnologias, já que é uma arquitetura voltada para sistemas de hipermídia distribuída onde a *Web* é somente uma instancia. Analisando

os fatos surgi outra dificuldade que seria como aproveitar as tecnologias da *Web* com as restrições do estilo REST. Diante de toda essa dificuldade surgi a ROA, uma arquitetura voltada para uma abordagem mais concreta para as dificuldades encontradas. Esta arquitetura segue as mesmas regras do REST e esta vinculada com as tecnologias da *Web* como HTTP, URI além dos formatos para transferência de dados como o XML, XHTML e JSON (POLONIA, 2011) e (RICHARDSON; RUBY, 2007).

Atualmente o número de organizações que fazem uso do REST para implementação de serviços *Web* teve um aumento significativo, devido aos serviços *RESTful Web Services* possuir um grau de complexidade bem menor, não exigir tanta habilidade e ter um custo de entrada menor que as plataformas similares. Mas vale lembrar que somente *RESTful Web Services* não fornece uma solução empresarial completa (SOAINSTITUTE.ORG...2013).

2.2. PRINCIPAIS CONCEITOS DA ARQUITETURA ORIENTADA A RECURSOS

A arquitetura orientada a recurso (ROA) e REST normalmente são usados no desenvolvimento de sistemas focados em recursos. Recursos é algo que um cliente tem interesse em obter de um serviço *Web* e este por sua vez tem a capacidade de fornecê-lo. Os recursos devem ter um nome e um endereço para trabalhar na *Web*, para isso é usado o *Universal Resource Identifier* (URI) contendo o nome de recursos e a localização dos mesmos, permitindo que este recurso torne-se visível a *Web* de tal forma que um cliente possa acessar a representação deste recurso, ou seja, “um recurso é tudo aquilo que deve ser acessado pelo cliente e transferido entre o mesmo e um servidor” (FRANÇA et al., 2011).

ROA é uma arquitetura baseadas em serviços Web e implementada a partir de tecnologias da *Web* como o protocolo HTTP e a linguagem de marcação XML. *A* arquitetura ROA também define algumas propriedades e conceitos, dentre estes conceitos, merecem destaque os recursos e os identificadores de recursos, já as

propriedades vale destacar a endereçabilidade, *stateless* (sem estado), conectividade e interface uniforme (SILVESTRE; POLÔNIA, 2008).

2.2.1. Recursos

O recurso é a parte mais importante da ROA sendo equivalente ao coração para o corpo humano. Segundo Polônia (2011, p. 54), “os recursos são definidos a partir de conceitos existentes no domínio da aplicação em questão”.

Um recurso é algo que podemos armazenar em um computador e representa-lo como uma sequencia de *bits*, como por exemplo, um documento, uma imagem entre outros. Este recurso pode ser um objeto físico, como por exemplo, uma maça ou um conceito abstrato como a coragem (RICHARDSON; RUBY, 2007).

Segue abaixo alguns recursos possíveis (RICHARDSON; RUBY, 2007):

- Um número primo qualquer;
- Informações sobre REST;
- O número primo entre 5 e 200;
- Uma relação entre dois amigos: João e Maria;
- Os números de vendas de baterias no mês de maio.

Podemos citar também uma coleção de recursos como, por exemplo, a coleção de todos os equipamentos de uma usina, onde mapearia um conjunto de enlaces, cada um apontando para o recurso de um equipamento. Outro exemplo é resultado de algoritmos expostos como recursos para buscar por um determinado equipamento, ou filtragem do histórico de aquisição de um equipamento. Também muito utilizado neste tipo de arquitetura é a exposição de uma coleção de recursos no ponto de entrada da aplicação, pelos quais um cliente pode navegar. O desenvolvedor também pode criar recursos conforme as suas necessidades, definindo diferentes graus de granularidade para um mesmo sistema e até mesmo unir recursos existentes (POLONIA, 2011).

2.2.2. Identificadores de recurso

Um identificador de recurso é o meio pelo qual se torna acessível um recurso, ou seja, é dado um nome e lugar onde seja possível encontra-lo. Sendo assim um recurso precisa de pelo menos um identificador para tornar-se acessível (POLONIA, 2011).

Segundo França et al. (2011, p. 111), “o princípio da identificação dos recursos esta relacionado ao uso de URIs que fornecem endereços únicos e globais para identificação de um recurso”. URIs permitem a navegação entre recursos que interagem entre si, visto que elas são utilizadas para indicar o escopo da informação, fornecendo meios para que haja a navegação, ou seja, um identificador pode indicar os sub-recursos pertinentes com um recurso em um dado momento. Por exemplo, o *path* do recurso “/equipamento/agregado” presente na requisição, agregado é um sub-recurso de equipamento (FRANÇA et al., 2011).

2.2.2.1. Uniform Resource Identifiers (URI)

A URI (*Uniform Resource Identifiers*) tem como propósito determinar que um determinado conceito do sistema seja um recurso e que este seja identificado para que haja um interação entre os diferentes componente interessados neste recurso. Cada URI é responsável por identificar um recurso apenas e ao mesmo tempo o torna endereçável, liberando para que ele seja manipulado. Um recurso exerce uma relação de um para muitos com uma URI, isto é, um recurso pode ser referenciado por muitas URIs, mas uma URI só pode ser referenciada a um recurso (POLONIA, 2011).

Como o objetivo da ROA é construir recursos que possam ser utilizados por pessoas e máquinas, as URIs devem ser associadas ao significado dos recursos apontados de forma intuitiva e estrutura, facilitando a manipulação por pessoas. Por exemplo,

para colher informações de equipamentos de uma usina, aconselha-se utilizar a seguinte URI: “/usina/equipamentos” (RICHARDSON, RUBY, 2007).

Seguindo a mesma linha do exemplo acima segue abaixo outros para frisar o conceito de URI:

- “/equipamentos” retorna uma lista de equipamentos;
- “/equipamentos/modelo” retorna uma lista de equipamentos de acordo com o modelo informado;
- “/equipamento/categoriaoperacional” retorna uma lista de equipamentos de acordo com a categoria operacional informada;
- “/equipamento/setor” retorna uma lista de equipamentos de acordo com o setor informado;
- “/equipamento/modalidade” retorna uma lista de equipamentos de acordo com a modalidade informada.

Vale lembrar que as URIs possuem algumas utilidades como por exemplo, ser utilizada para oferecer integração entre recursos sem a necessidade do uso do corpo da resposta e no contexto da *Web* possibilita a utilização de *links* para recursos, os quais podem ser utilizados identificadores bem conhecidos (FRANÇA et al., 2011).

2.3. PROPRIEDADES

Nesta seção será mostrada a estrutura e a forma de expor a arquitetura RESTful concreta, ou seja, a arquitetura orientada a recursos (ROA). Será apresentado como a ROA transforma um problema em serviços RESTful, utilizando arranjos de URIs, HTTP e XML, funcionando como os demais serviços *Web* e facilitando o trabalho árduo dos desenvolvedores. Também serão apresentadas as partes móveis da ROA como os recursos, seus nomes, suas representações e as ligações existentes entre eles (RICHARDSON; RUBY, 2007).

Vale lembrar que a aplicação que adota a arquitetura ROA deve necessária possuir as propriedades que serão descritas nesta seção, como endereçabilidade, *stateless* (sem estado), conectividade e interface uniforme. (POLÔNIA, 2011).

2.3.1. Endereçabilidade

Para exibir um conjunto de informações importantes é necessário ter aplicações endereçáveis para expô-las como recursos na *Web*. Estes recursos são atribuídos a URIs, tornando a aplicação endereçável, ou seja, indicando o endereço onde as informações estão contidas. Portanto pessoas que estão navegando pela Web, podem localizar estes recursos através do preenchimento, por exemplo, de um campo de busca com a expressão desejada. Esta por sua vez é convertida em URI que contém as informações necessárias para que a busca retorne conforme solicitado pelo usuário (RICHARDSON, RUBY, 2007).

Esta propriedade é extremamente importante para qualquer site ou aplicação *Web*, pois sem o endereçamento os sites conhecidos que usamos diariamente não seriam facilmente encontrados. Por exemplo, para encontrar informações sobre “futebol”, terá que ser digitado no *browser* a URI: http://www.google.com.br/search?q=futebol. Se o mecanismo da *Google* não fosse uma aplicação endereçável ou se o protocolo HTTP não fosse endereçável, não seria possível usar a URI acima, seria necessário efetuar uma solicitação ao site da *Google* e em seguida informar o assunto a ser pesquisado (LIMA, 2012).

O endereçamento também permite fazer proveito deste mesmo URI para retornar exatamente uma pesquisa feita em outro momento ou também ser repassada para outra pessoa que ela terá o mesmo resultado desta pesquisa, ou seja, quando chega uma requisição repetida, podemos enviar um resultado salvo anteriormente ao invés de consumir recursos de banda para realizar a mesma requisição duas vezes. Graças à endereçabilidade é possível saber quais requisições já foram efetuadas (RICHARDSON, RUBY, 2007).

2.3.2. Stateless

A propriedade *Stateless* ou sem estado é proveniente do estilo arquitetural REST. O objetivo da ROA determina que se faça o uso correto do protocolo HTTP, portanto uma aplicação sem estado, cada acesso representa um acesso a um recurso, não importando quais recursos o cliente já visitou e muitos menos a ordem de acesso deles, caso exista interesse em guardar esse tipo de informação, deverá ser guardada pelo cliente. Este fato torna a aplicação livre de complexidade adicional, visto que o servidor teria que analisar o histórico de requisições para tomar as decisões cabíveis, fazendo com que o processo ficasse mais complicado e consequentemente prejudicando o desempenho, visto que o servidor teria que guardar as informações de cada cliente (FRANÇA et al., 2011).

A propriedade sem estado denota que cada requisição HTTP é realizada em completo isolamento, ou seja, quando um cliente envia uma requisição HTTP, vai incluso toda informação imprescindível para que o servidor atenda a requisição, nunca fazendo uso de informações de requisições anteriores (SILVESTRE; POLÔNIA, 2008).

Na aplicação sem estado o cliente faz uma solicitação e termina onde começou, ou seja, cada solicitação é totalmente independente uma das outras, podendo o cliente solicitar determinados recursos várias vezes em qualquer ordem (LIMA, 2012).

A Figura 1 abaixo representa um diagrama de estado demonstrando como é uma solicitação efetuada a uma aplicação sem estado. Observe na figura abaixo que não a qualquer exigência para seguir uma sequência de passos para alcançar o objetivo, deixando o cliente à vontade para escolher o caminho desejado, ou seja, o cliente se optar, pode ir direto ao objetivo, não sendo necessariamente obrigatório seguir determinados passos para chegar ao objetivo desejado.

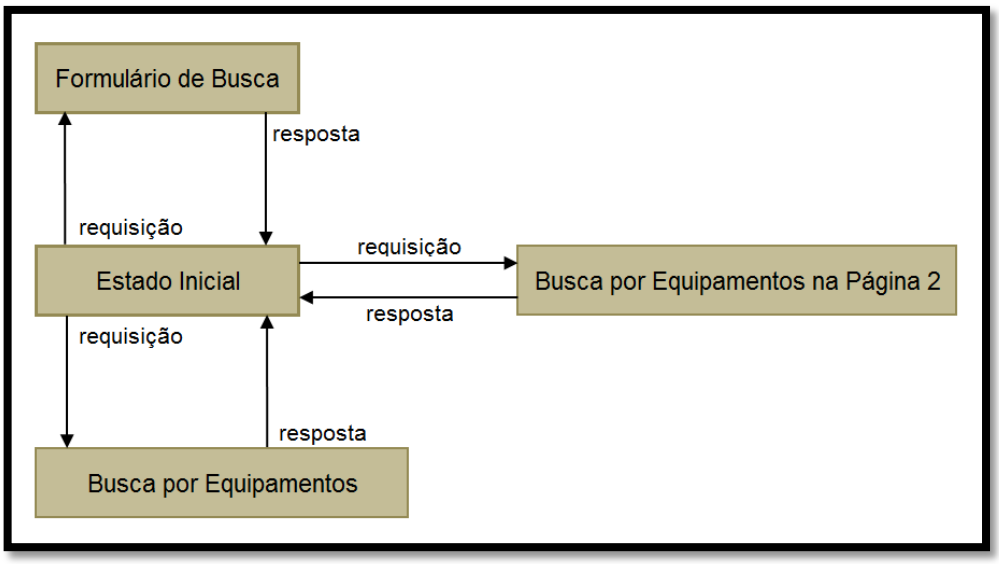

Figura 1 – Aplicação Sem Estado (In: SILVESTRE; POLÔNIA, 2008, p. 32)

Em uma aplicação com estado, o cliente deve seguir cronologicamente uma ordem, ou seja, para que o cliente chegue a pagina dois, primeiramente ele deve passar pela página um. A Figura 2 mostra como é os passos de uma aplicação com estado, observe que o cliente deve seguir uma sequência de passos obrigatórios para atingir um determinado objetivo, não existindo a possibilidade de ir direto ao objetivo desejado (LIMA, 2012).

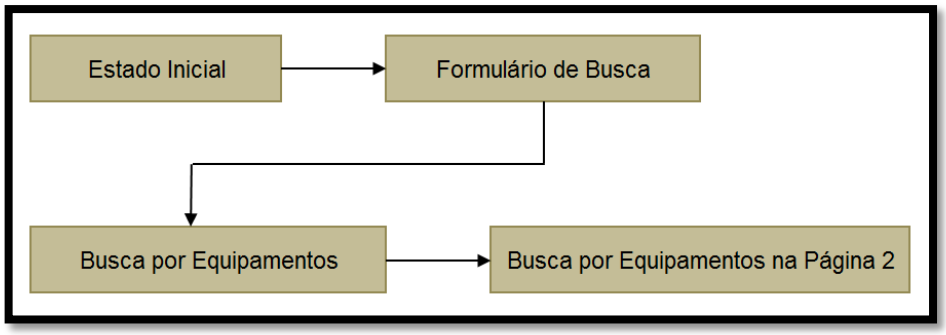

Figura 2 – Aplicação com Estado (In: SILVESTRE; POLÔNIA, 2008, p. 32)

Uma das vantagens de se trabalhar com aplicação sem estado, esta no fato de permitir que seja recuperado um diretório sobre “futebol” até a página dez, por

exemplo, e voltar várias vezes para a última página sem a necessidade de passar pelas páginas anteriores (RICHARDSON, RUBY, 2007).

2.3.3. Conectividade

As representações de um determinado recurso podem ser aproveitadas para direcionar para outros recursos relacionados com o primeiro recurso acessado. Essa forma faz com que o cliente escolha por qual caminho deseja seguir, indicando para qual estado da aplicação deseja ir. A conectividade garante a aplicação orientada a recursos, formar uma estrutura capaz ligar a outras aplicações e até mesmo a recursos diversos da *Web* (POLÔNIA, 2011).

Segundo Silvestre; Polônia (2008, p. 33), “conectividade é a propriedade que permite que um cliente navegue por entre os recursos da aplicação através de enlaces hipermídia embutidos nas representações”. Estas representações com enlaces são enviadas pelo servidor para o cliente com um conjunto de URIs com possíveis estados da aplicação (SILVESTRE; POLÔNIA, 2008).

A Figura 3 mostra a representação gráfica da estrutura da propriedade conectividade em uma aplicação conexa, ou seja, interligadas entre si.

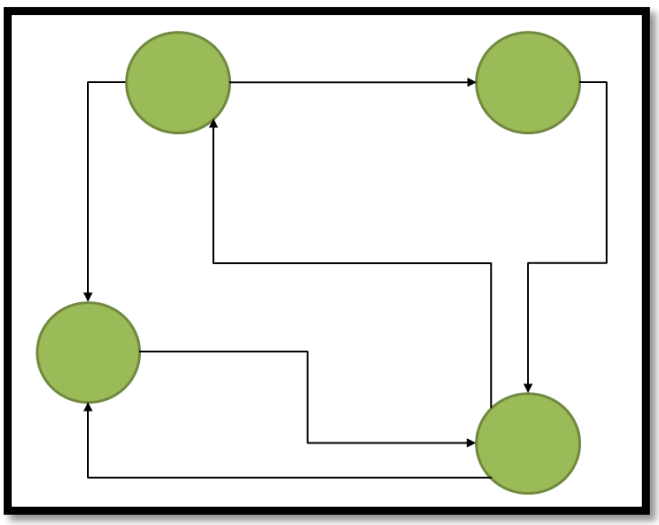

Figura 3 – Aplicação Conexa (In: SILVESTRE; POLÔNIA, 2008, p. 34)

Quando efetuamos uma pesquisa em um site de busca como do *Google*, o servidor deles disponibiliza uma cadeia de representações para o assunto que esta sendo pesquisado. Estas representações são chamadas de hipermídia e possuem além de dados, *links* para outros recursos, propiciando ao cliente que ele faça a escolha do caminho que deseja seguir (RICHARDSON, RUBY, 2007).

2.3.4. Interface uniforme

Na arquitetura REST é especificada uma interface uniforme, mas em geral não é especificado como deve ser feito. Na ROA esta interface é oriunda do protocolo HTTP. Esta interface permite acessar e manipular os recursos, fazendo com que seja criada uma linguagem comum, onde é possível interagir com qualquer recurso na *Web* (FRANÇA et al., 2011).

Os métodos HTTP utilizados na ROA são (RICHARDSON; RUBY, 2007):

- GET: usado para obter uma representação de um recurso;
- PUT: utilizado para criar um novo recurso ou modificar um já existente;
- POST: cria um novo recurso;
- DELETE: remove um recurso existente;
- HEAD: utilizado para obter metadados sobre um recurso;
- OPTIONS: obtém quais métodos são suportados por um recurso.

Acima podemos observar que os métodos PUT e POST embora sejam parecidos, existe uma pequena diferença. Esta diferença esta basicamente na definição da URI mencionada pelo cliente na requisição. No POST a URI representa um ponto onde será adicionada a requisição, não sendo necessário indicar a URI onde será criado ou atualizado o recurso, visto que esta missão fica por conta do servidor. No PUT por sua vez, o cliente determina obrigatoriamente na URI, o caminho onde deseja

que a requisição esteja após o processamento da mesma pelo servidor (Polônia, 2011).

Neste capítulo ficou explicito que a ROA surgiu para definir uma abordagem mais concreta para desenvolvimento de serviços *Web* agregando os princípios de REST com as tecnologias da *Web* e seus formatos para transferência de dados, tornando mais fácil e definido o desenvolvimento de serviços *Web*. ROA faz com que a aplicação seja feita em menor tempo, consequentemente diminuindo o custo, visto que o desenvolvedor fica focado somente no que interessa, deixando de lado o restante que é comum a todas as aplicações.

Foram abordados também os principais conceitos e propriedades necessários para que uma arquitetura seja considerada orientada a recursos (ROA), apresentando as definições e a importância de cada conceito e propriedade dentro da ROA, além de exemplificar o funcionamento de cada item.

3 – PLATAFORMA JAVA ENTERPRISE EDITION (JEE)

No mundo globalizado e dinâmico em que vivemos hoje, cada vez mais aumenta a demanda por aplicações distribuídas, portáteis, que aproveite a velocidade, segurança e confiabilidade das tecnologias existentes nos servidores. As aplicações corporativas fornecem geralmente a lógica dos negócios das empresas e muitas vezes estas aplicações interagem com outros softwares da empresa. No entanto no mundo tecnológico atual, é necessário que as aplicações corporativas sejam idealizadas, desenvolvidas com custo baixo, no menor tempo possível e com menos recursos. A plataforma *Java Enterprise Edition* (Java EE) surgiu com a proposta de tornar mais fácil e rápido esta interação entre aplicações distintas, fornecendo em sua plataforma um poderoso conjunto de recursos capazes de reduzir o tempo de desenvolvimento, a complexidade da aplicação e melhorar o desempenho dos aplicativos (ORACLE, 2013).

As aplicações *Web* existentes nos dias atuais possuem regras de negócio muito complexas, exigindo do desenvolvedor um grande trabalho para codificar todas estas regras. Além dessas regras chamadas de requisitos funcionais de uma aplicação, temos que nos preocupar com os requisitos não funcionais que a nossa aplicação tem que prover tais como, persistência em banco de dados, transação, acesso remoto, *Web Services*, gerenciamento de *threads*, gerenciamento de conexões HTTP, cache de objetos, gerenciamento da sessão *Web*, balanceamento de carga, entre outras. Diante dessa dificuldade a *Sun* que agora foi comprada pela *Oracle* criou um conjunto de especificações sobre a plataforma *Java Standard Edition* (JSE) para facilitar a vida do desenvolvedor (CAELUM, 2012).

2.1. JAVA ENTERPRISE EDITION (JEE)

Java é uma linguagem de desenvolvimento de aplicações em geral, integrada com a *Web* e que dá suporte ao desenvolvimento de aplicações em larga escala. É uma linguagem orientada a objetos, de alto nível, robusta, segura, bem estruturada,

distribuída e permite o desenvolvimento de aplicações para várias plataformas e sistemas operacionais. Desde o seu lançamento o Java vem sofrendo aprimoramentos e consequentemente aumentando o número de aplicações, assim como o número de bibliotecas padrões, onde acabou levando ao surgimento de três plataformas: *Java Standard Edition* (JSE), *Java Enterprise Edition* (JEE) e o *Java Micro Edition* (JME), sendo este último ambiente de desenvolvimento voltado para dispositivos móveis e portáteis (BROEMMER, 2003).

O Java SE é um ambiente de desenvolvimento voltado para *desktops* e servidores, além de ser a base principal para as demais plataformas (JEE e JME), ou seja, esses ambientes de desenvolvimento são versões aperfeiçoadas para as aplicações a que se propõe. Enfim o *Java Enterprise Edition* (JEE) é uma plataforma voltada para redes, *Internet*, intranets entre outros. Esta plataforma contém recursos desenvolvidos especialmente para acesso a servidores, sistemas de e-mail, banco de dados, entre outros, conseguindo com estas características suportar vários usuários simultaneamente (GONÇALVES, 2010).

Java EE é uma plataforma que contém um conjunto de tecnologias que reduz significativamente o custo e a complexidade do desenvolvimento, fazendo com que o desenvolvedor escreva menos código e consequentemente diminua o tempo de desenvolvimento, os riscos do projeto e até mesmo os problemas com manutenção, aproveitando toda infraestrutura já existente na plataforma para desenvolvimento de aplicações corporativas distribuídas (SAMPAIO, 2011).

Com a padronização da plataforma tornou-se possível também resolver o problema de migração de fornecedores, caso seja necessário trocar a implementação por outra mais rápida e que use menos memória não será preciso alterar o software existente, visto que o Java EE possui uma especificação muito bem definida. O que deverá ser mudado é apenas a implementação da especificação tornando o software livre, ou seja, não fica preso a um código já que a especificação garante que a aplicação funciona com implementação de outro fornecedor (GONÇALVES, 2010).

2.2. CONTAINERS

Containers são interfaces entre um componente e a funcionalidade exclusiva da plataforma de baixo nível suportando um componente. Para executar uma aplicação *Web*, *enterprise bean*, ou algum componente do aplicativo cliente, primeiro deve ser disposto em um módulo Java EE e implantado em seu recipiente (JENDROCK et al., 2013).

O *Java Enterprise Edition* possui um modelo de programação baseado em containers, estes por sua vez contêm vários recursos necessários para desenvolvimento de aplicações corporativas. Com todos estes recursos o desenvolvedor tem um grande ganho na escrita do código, pois escreve menos, consequentemente diminui o tempo de desenvolvimento, os riscos e os problemas relacionados com manutenção (SAMPAIO, 2011).

O modelo de *containers* fornece um ambiente estruturado para desenvolvimento de aplicações corporativas fazendo com o desenvolvedor se preocupe somente com as funcionalidades desejadas, deixando o básico para os serviços e APIs dos *containers* (JENDROCK et al., 2013).

Segundo Sampaio (2011, p. 20):

> Cada tipo de aplicação prevista no Java EE possui um tipo de container especifico. Aplicações servidoras web rodam em um Container Web. Componentes remotos rodam em um Container EJB. E aplicações clientes que utilizam componentes JavaEE rodam no Application Client Container.

A *Java Enterprise Edition* é um modelo de programação fundamentado em *containers*, onde são fornecidos todos os serviços indispensáveis para o desenvolvimento de uma aplicação distribuída. Os componentes de aplicação

contidos no *container* do Java EE são (OLIVEIRA, 2013) e (JENDROCK et al., 2013):

- ***Java EE Server:*** É a parcela de tempo de execução de um aplicativo Java EE. O Java EE provê *containers web* e EJB.
- ***Enterprise JavaBeans* (EJB):** este componente roda no Java EE e é um dos *containers* mais conhecidos da arquitetura. Define um ambiente de execução de aplicações corporativas, tais como gerenciamento de ciclo de vida, concorrência, serviços de segurança, transação, entre outros.
- ***Web container*:** aborda a implementação de interação de componente *Web* com a arquitetura Java EE, provendo um ambiente de execução para componentes do tipo *Web* que abrange desde segurança até gerenciamento de vida e transação. Inclui os *servelets* que nada mais é do que classes Java que podem ser carregadas e executas em um servidor *Web* e disponibilizadas através de serviços de rede relativos aos pedidos e respostas enviados com o protocolo HTTP. Também possui o *JavaServer Pages* (JSP) que é uma tecnologia *Web* que utiliza, por exemplo, na linguagem de *scripting* e objetos Java no servidor para retornar a um cliente de forma dinâmica.
- ***Application client**:* gerencia e aceita a execução de componentes do cliente da aplicação;
- ***Applet container*:** gerencia a execução de *applet*, ou seja, consiste em um navegador *Web* e *Java Plug-in* em execução simultânea no cliente.

Podemos observar na Figura 4 os containers e componentes descritos acima, sendo que na primeira camada temos o *browser* onde também pode ter aplicações Java executando dentro da aplicação *client container*. Na camada Java EE Server temos os elementos *Web* da arquitetura Java EE tais como os *JavaServer Faces* (JSF), os *Servelets* e o *JavaServer Pages* (JPA). Na camada EJB *container* estão os elementos remotos onde são implementadas as regras de negócios da aplicação e por último temos os servidores de banco de dados ou Mainframes (SAMPAIO, 2011).

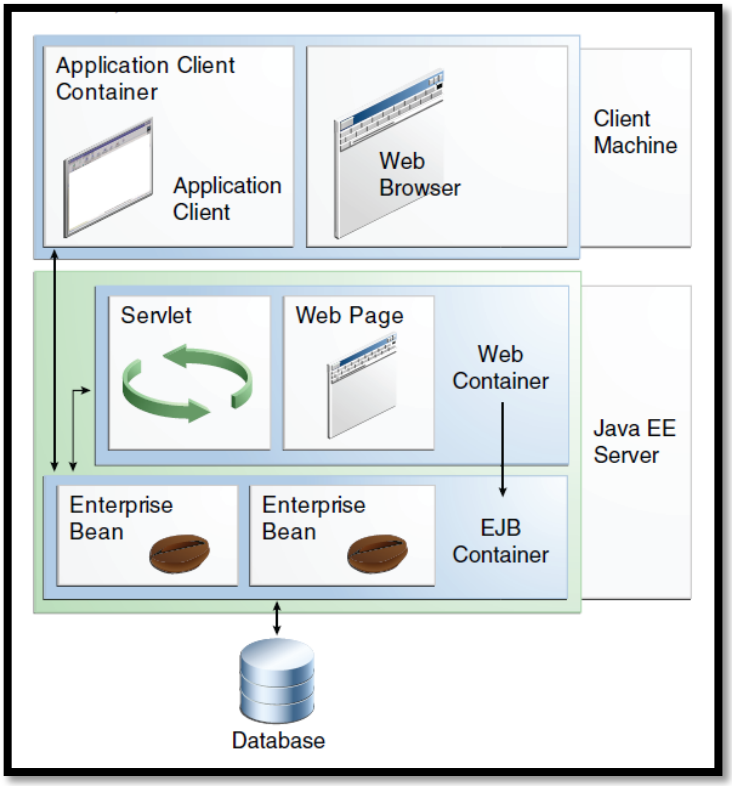

Figura 4 – Java EE *Server* e *Containers* (In: JENDROCK et al., 2013, p. 48)

Um *container* do Java EE contém todas as APIs e serviços da especificação da plataforma e para criar e executar aplicações desenvolvidas no Java EE precisamos de *containers,* sendo que estes podem ser voltados basicamente para *Web*, como é o caso do *Tomcat* ou completo como o *Geronimo* (SAMPAIO, 2011).

2.3. PRINCIPAIS RECURSOS DA PLATAFORMA JEE

A plataforma JEE fornece um conjunto de funcionalidades para o desenvolvimento de aplicações servidoras capazes de suportar o desenvolvimento de aplicações corporativas. Esta característica é garantida devido ao conjunto de serviços, interfaces de programação, de aplicação (APIs) e protocolos que proporcionam diversas funcionalidades para desenvolvimento de aplicações multicamadas focadas em serviços *Web* (OLIVEIRA, 2013).

O Java EE é uma plataforma que roda sob o Java SE, porém possui um conjunto de recursos capazes de fazer o mapeamento objeto-relacional, arquiteturas multicamada, distribuídas e *Web Services*. Abaixo serão descritos as principais APIs e recursos do Java EE tais (GONÇALVES, 2010):

- ***Enterprise* Java *Beans* (EJB):** são códigos com campos e métodos para programar os módulos de lógica de negócios. Um *bean* corporativo é um grupo onde pode ser usado sozinho ou com outros *beans* existentes na empresa para executar a lógica de negócio em um servidor JavaEE. Os *beans* corporativos podem ser de sessão ou de mensagem *driven*. Os de sessão monta uma interação transiente com um cliente, sendo que quando o cliente encerra a execução o bean de sessão e os dados são apagados, já a mensagem *driven* alia propriedades de um *bean* de sessão e um ouvinte de mensagem, possibilitando que um componente de negócio receba mensagens de forma assíncrona. Estes serviços normalmente são serviços de mensagens *Java Message* (JMS). De uma forma geral é uma arquitetura de componentes responsáveis por simplificar o desenvolvimento de aplicações distribuídas, transacionais, seguras e portáteis (JENDROCK et al., 2013) e (OLIVEIRA, 2013).
- **Java API for XML *Parsing* (JAXP):** esta API pertence à plataforma do Java SE e fornece suporte para análise sintática de documentos XML e fornece analisadores padrões como o SAX e DOM e o XSLT, tendo assim um maior controle sobre a apresentação de dados e capacidade de conversão de dados. Essas bibliotecas também contêm outras APIs exclusivas para uso de XML e *Web Services*, como a JAX-RPC ou JAXR. Fornece também suporte para trabalhar com esquemas que poderia gerar algum tipo de conflito de nomes (OLIVEIRA, 2013).
- **Java *Architecture* for XML *Binding* (JAXB):** fornece um modo adequado de ligar um esquema XML para a execução em aplicações Java. Tem a capacidade de mapear a classe objeto em XML e vice versa, ou seja, permite armazenar e recuperar os dados na memória em qualquer formato XML.

JAXB pode ser usada junto com a JAX-WS ou de forma independente (JENDROCK et al., 2010).

- **Java *Database Connectivity* (JDBC):** esta API permite a utilização de comandos SQL em métodos da linguagem de programação Java. A JDBC também pode ser usada para acessar diversos tipos de bancos de dados usando a mesma API. Outra forma de usar esta API é a partir de um servlet ou uma página JSP para ter acesso direto ao banco de dados, não sendo necessário passar por um *bean* corporativo. A JDBC possui duas partes: uma interface em nível de aplicação usando componentes da aplicação para acesso ao banco de dados e outra interface do provedor de serviço responsável por alojar um *driver* JDBC para a plataforma Java EE (JENDROCK et al., 2013).
- **Java *Naming and Directory Interface* (JNDI):** tem como objetivo fornecer aplicações com uma interface padrão para localização de usuários, máquinas, objetos, redes e serviços. A JNDI pode ser usada tanto para localizar um computador em rede, quanto buscar objetos Java. Usando esta API, uma aplicação Java EE pode armazenar e recuperar qualquer tipo de objeto e assim permitir que aplicativos Java possam ser executados simultaneamente com outras aplicações e sistemas legados (JENDROCK et al., 2010).
- **Java *Persistence* API (JPA):** é um recurso baseado em padrões Java para persistência de dados. Esta persistência emprega um objeto com abordagem relacional para ocupar o espaço entre um modelo orientado a objeto e um banco de dados relacional, ou seja, padroniza o acesso ao banco de dados por meio de persistência e mapeamento do objeto-relacional das classes Java (JENDROCK et al., 2013).
- **Java*Server Pages* (JSP):** possibilita a colocação de trechos de código *servlet* diretamente em um documento baseado em texto como a página JSP e esta por sua vez contém dois tipos de texto, sendo um de dados estáticos, que pode ser escritas em qualquer formato baseado em texto, como o HTML ou XML e elementos JSP que define o modo de construção do conteúdo

dinâmico da página. De um modo geral esta tecnologia facilita o desenvolvimento de conteúdo dinâmico *Web* (OLIVEIRA, 2013).

- ***Servelets* Java:** esta tecnologia permite a definição de classes de *servlet* HTTP exclusivos. Esta classe estende as competências dos servidores capazes de hospedar aplicativos acessados por meio de um modelo de programação de pedido e resposta. Os *servelets* pode receber qualquer tipo de pedido, mas são habitualmente usados para estender os aplicativos hospedados por servidores *Web*, ou seja, fornece uma estrutura simples e sólida para estender a funcionalidade de um servidor *Web* para acessar sistemas corporativos (JENDROCK et al., 2010).
- **SOAP *with Attachments* API for Java (SAAJ):** é uma API de baixo nível e que permite a criação e envio de mensagens com especificação SOAP. A maioria dos desenvolvedores prefere usar a JAX-WS que trabalha em um nível mais alto do que usar a API SAAJ que roda em baixo nível (JENDROCK et al., 2013).
- **Java API for *RESTful Web Services* (JAX-RS):** determina APIs para o desenvolvimento de *Web Services* empregando a arquitetura *Representational State Transfer* (REST). A aplicação JAX-RS consiste em uma aplicação *Web* empacotada como um *servlet* juntamente com bibliotecas necessárias (JENDROCK et al., 2010).

A Figura 5 ilustra todas as APIs e recursos descritos acima mostrando a localização de cada. Podemos observar também os segmentos dos Java *Enterprise Edition* com seus componentes contidos nos *containers*, onde podem ser chamados através de diferentes protocolos (GONÇALVES, 2010).

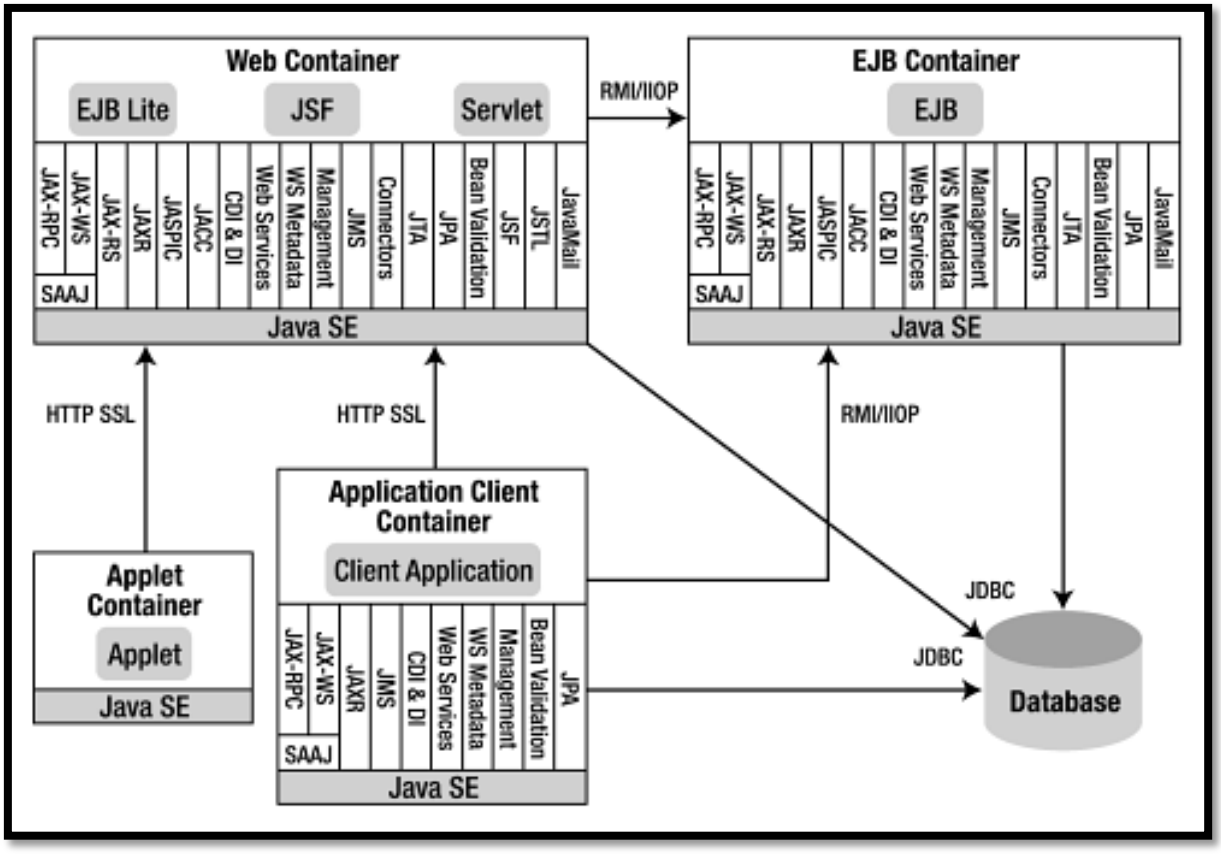

Figura 5 – Componentes do JEE por *Containers* (In: GONÇALVES, 2010, p. 8)

Depois de ter visto a descrição das APIs e recursos e observado a Figura 5, fica claro que o Java EE possui um conjunto de especificações bem detalhadas, para que quando implementadas, possam ajudam os desenvolvedores a tirar proveito e reutilizar toda essa infraestrutura existente nesta plataforma (CAELUM, 2012).

2.4. SERVIDOR DE APLICAÇÃO JAVA EE

O servidor de aplicação nada mais é do que uma aplicação desenvolvida em Java que fornece uma infraestrutura de serviços para executar aplicações distribuídas. Eles são executados em servidores e acessados pelo cliente através de uma conexão de rede. Também possuem muitas vantagens em relação ao modelo cliente/servidor, visto que ele fornece uma plataforma muito rica para facilitar o desenvolvimento, visto que é priorizado o compartilhamento de componentes e

aplicações facilitando o desenvolvimento, a manutenção e o gerenciamento de aplicações complexas (RENOUF, 2009).

Os servidores de aplicações utilizam uma arquitetura chamada de três camadas ou n-camadas permitindo um melhor aproveitamento das propriedades de cada componente (servidor de banco de dados, servidor de aplicação e cliente). A primeira camada também chamada de *front-end* são os *browsers*, a segunda camada é a aplicação propriamente dita sendo executada no servidor de aplicação e a terceira é o servidor de banco de dados (RENOUF, 2009).

Além das características já citadas anteriormente, existem também outros serviços como:

- **Tolerância a falhas:** possui recuperação e distribuição de componentes em clones dos servidores.
- **Balanceamento de cargas:** permite a distribuição de clientes para maximizar a utilização dos recursos promovendo desta forma um ganho de desempenho.
- **Gerenciamento de componentes:** permite a manipulação de componentes e serviços como notificação, gerenciamento de sessão e distribuição lógica de negócios, garantido um aumento de desempenho.
- **Gerenciamento de transações:** asseguram a integridade da transação entre os aplicativos e os diversos bancos de dados envolvidos.
- **Console de gerenciamento:** com um único sistema gráfico é possível gerenciar vários servidores de aplicação.
- **Segurança:** garante a segurança da aplicação.

Existem vários servidores de aplicação desenvolvido em Java, dentre os principais podemos citar:

- *JBoss (RedHat)*, gratuito;
- *GlassFish (Oracle)*, gratuito;
- *Geronimo (Apache)*, gratuito;

- *WebSphere* (IBM).

Na plataforma Java também tem os servidores que implementam somente as especificações para uso de aplicações *Web*, dentre os principais podemos destacar o *Tomcat* (*Apache*) e o *Jetty* (*Codehaus*) ambos gratuitos.

Hoje no mundo globalizado e concorrido às empresas fica cada dia mais dependente de aplicações corporativas para administrar seus negócios. Neste capítulo podemos perceber o quanto as funcionalidades do Java EE são fundamentais para os dias atuais, devido à praticidade que esta plataforma oferece aos desenvolvedores.

Os recursos e APIs ajudam no desenvolvimento das aplicações corporativas, visto que eles diminuem o trabalho que é comum à maioria das aplicações, fazendo que o desenvolvedor fique mais focado nas regras de negócio da aplicação ganhando em tempo, custo e manutenção, ou seja, tudo o que as empresas precisam para se manter no mercado fortemente concorrido dos dias atuais.

3 – WEB SERVICES

No final dos anos 90, estavam sendo desenvolvidos vários sistemas baseados em HTTP e XML, mas cada sistema com suas características próprias de implementar segurança, confiabilidade, gerenciamento de transações. Este cenário deu início a um caos devido à incompatibilidade entre sistemas, dificuldade na manutenção e em contrapartida surgiu à necessidade de elaborar uma maneira de realizar estas tarefas, foi então que surgiu o conceito de *Web Services* (MORO; DORNELES; REBONATTO, 2011).

Web Services surgiu para efetuar a comunicação de dados entre aplicativos desenvolvidos em diferentes plataformas, através dos protocolos da *Internet.* Este processo consiste na transferência de dados realizados por aplicativos através da *Internet* entre empresas e pessoas sem que haja diretamente a interferência humana. Após o surgimento destas especificações, as grandes empresas como IBM, *Microsoft*, entre outras, entraram em acordo para dar suporte às especificações de comunicação entre aplicativos desenvolvidos em plataforma diferentes, ou seja, integrar seus sistemas ao invés de ficar desenvolvendo soluções parecidas com abordagens divergentes como vinha sendo praticado naquela época (AVELAREDUARTE, 2013) e (CERAMI, 2002).

No decorrer dos anos a tecnologia vai sofrendo várias mudanças e estas por sua vez implicam em novas necessidades, dentre estas podemos destacar a unificação entre aplicações, a utilização integrada de processos em diferentes aplicações e escrita em diferentes linguagens. Diversas vezes é necessário que as informações de uma organização sejam acessadas diretamente por outra sem a intervenção humana como é o caso da nota fiscal eletrônica, onde as empresas podem emitir as notas fiscais eletronicamente através de uma aplicação fornecida pelo Ministério da Fazenda. Diante de todas estas questões surgi à tecnologia *Web Services*, criada com o intuito de disponibilizar formas para unificar aplicações distintas e capacitar a integração e consumo de informações. *Web Service* pode trazer para empresas algumas vantagens como agilidade para os processos e eficiência na comunicação,

pois a transferência de dados entre aplicações passa a ser dinâmica e segura por não haver intervenção humana (MORO; DORNELES; REBONATTO, 2011).

4.1. DEFINIÇÃO CONCEITUAL

Web Services é um serviço oferecido por uma aplicação onde pode ser acessado por outras aplicações desenvolvidas em qualquer plataforma por meio de uma rede como a *Internet* (ALONSO et al., 2004).

Também pode ser definida também como sendo qualquer serviço na *Internet* que faz uso do padrão XML para efetuar troca de dados e não esteja necessariamente, vinculado a nenhum sistema operacional ou linguagem de programação exclusiva (CERAMI, 2002).

A Figura 6 ilustra a transferência de dados entre plataformas distintas, onde o computador A utiliza a linguagem *Perl* com o sistema operacional *Windows Sete* enquanto o computador B utiliza a linguagem Java com o sistema operacional *Linux*.

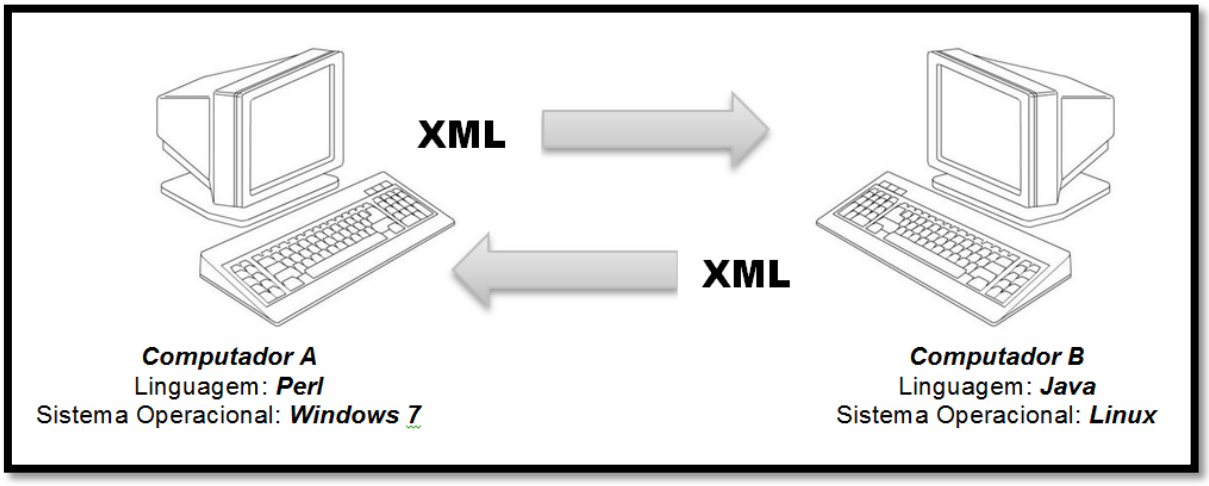

Figura 6 – Serviço básico de *Web Service* (In: CERAMI, 2002, p. 4)

Os *Web Services* são elementos que permitem que aplicações desenvolvidas em diversas plataformas enviem e recebam informações em diversos formatos, onde

cada aplicação pode ser desenvolvida na sua própria linguagem e representada para uma linguagem universal, como XML (MORO; DORNELES; REBONATTO, 2011).

Atualmente existem outros formatos de dados disponíveis na plataforma *Web* tais como, o HTML, JSON RSS, *Atom*, CSV entre outros, sendo que os mais utilizados são o XML e o JSON (SOUZA, 2010).

Os *Web Services* pode ser considerada também como interfaces para funcionalidades de aplicativos *Web*, que faz uso dos padrões de *Internet*, ou seja, aplicativos que podem ser acessados por meio de uma rede, utilizando-se de protocolos como HTTP ou XMPP (*Extensible Messaging and Presence Protocol)*. A Figura 7 ilustra o funcionamento de *Web Services* utilizando interfaces como descrito anteriormente (SOUZA, 2010).

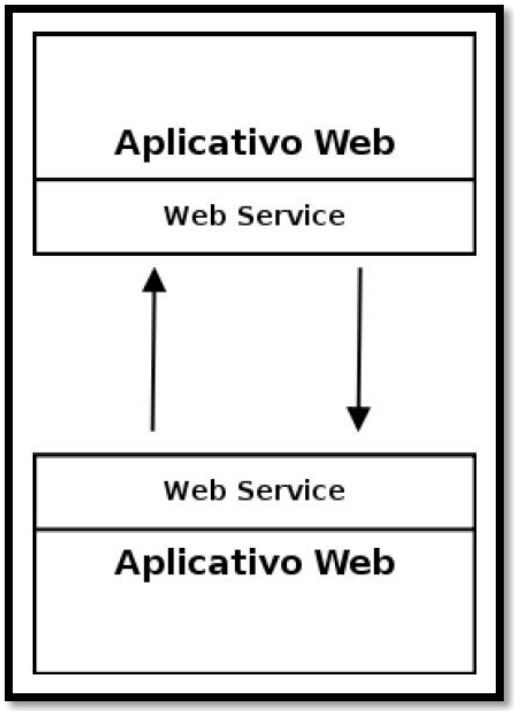

Figura 7 – Exemplo de *Web Services* com Interfaces (In: SOUZA, 2010, p. 13)

Uma característica importante da arquitetura *Web Services* é a possibilidade de utilizar diferentes formas de transmissão de informações pela rede utilizando

protocolos tais como HTTP, SMTP, FTP, RMI/IIOP ou até mesmo protocolos de mensagem proprietários (CERAMI, 2002).

A tecnologia dos *Web Services* fez com que a *Internet* torna-se uma plataforma voltada à aplicação e não voltado exclusivamente a seres humanos, visto que os dados são transmitidos entre aplicações sem a intervenção humana (SOUZA, 2010).

4.2. WEB SERVICE EM JAVA

A principal característica na tecnologia *Web Services* diz respeito à ideia de interoperabilidade, onde deva permitir que qualquer computador, programa, plataforma de hardware ou sistema operacional que estejam nos padrões determinados na *Internet*, como HTTP, SOAP e XML, possam interagir com o *Web Services* disponibilizado. O Java disponibiliza vários mecanismos através de API, incorporados na plataforma Java *Enterprise Edition* (ABINADER; LINS, 2006).

Na plataforma Java, os serviços *Web* podem ser implementados de várias maneiras. Nesta seção vamos descrever as funcionalidades das APIs “Java API for XML *Web Services* (JAX-WS)” e a “Java API *for RESTful Web Services* (JAX-RS)”, que fazem parte do conjunto de especificações da plataforma *Java Enterprise Edition* (JEE).

A API Java API for XML *Web Services* (JAX-WS) é uma tecnologia usada para a construção de serviços *Web* e clientes comunicando-se através do formato XML. Esta API também possibilita aos desenvolvedores à implementação orientado a mensagem, assim como serviços *Web Remote Procedure Call-oriented* (RPC-oriented). A JAX-WS invoca serviços de operações *Web* representada por um protocolo baseado em XML, como o SOAP. Esta especificação por sua vez define uma estrutura de envelopamento, regras de codificação e acordos para montar chamadas de serviços *Web* e respostas, transmitidos através de mensagens SOAP (arquivos XML) sobre o protocolo de *Internet* HTTP (JENDROCK et al., 2013).

A Figura 8 ilustra como é realizada a comunicação entre um *Web Services* construído em JAX-WS e um cliente.

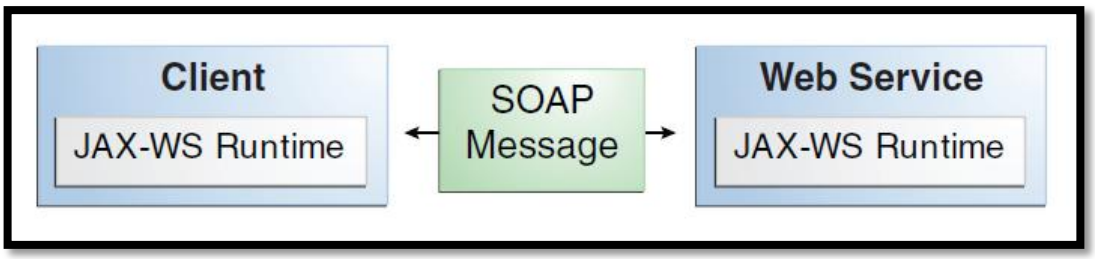

Figura 8 – Comunicação entre um service *Web* JAX-WS e um cliente (In: JENDROCK et al., 2013, p. 368)

As mensagens SOAP são bem complexas, porém a JAX-WS oculta toda essa complexidade do desenvolvedor da aplicação. O desenvolvedor especifica as operações de serviços *Web* no lado o servidor, definindo os métodos e interfaces através da linguagem de programação Java. Na aplicação cliente cria-se um proxy, ou seja, um objeto local para representar um serviço, para depois invocar os métodos no proxy. É importante salientar que com o JAX-WS, o desenvolvedor não gera e nem analisa as mensagens SOAP, quem faz a conversão das chamadas e resposta de API para as mensagens SOAP é o sistema de *runtime* da JAX-WS. A JAX-WS possui uma grande vantagem que é a flexibilidade de interagir com diversas plataformas, visto que esta API segue as especificações dos padrões da W3C, que é organização que define os padrões de funcionamento de um *Web Services* (JENDROCK et al., 2013).

A API Java API *for RESTful Web Services* (JAX-RS) é uma linguagem de programação projetada para simplificar o desenvolvimento das aplicações que fazem uso da arquitetura *Representational State Transfer (*REST). A JAX-RS utiliza anotações da linguagem Java para simplificar o desenvolvimento de serviços *Web RESTful*. As classes de linguagem de programação Java são usadas por meio de anotações JAX-RS inseridas na mesma para definir recursos e ações que podem ser executadas sobre estes recursos, sendo que estas anotações JAX-RS rodam em tempo de execução, gerando classes auxiliares e componentes para o recurso. Toda aplicação construída na plataforma Java EE possui classes de recurso JAX-RS, onde terão seus recursos configurados, as classes auxiliares e componentes

gerados e os recursos exibidos aos clientes através de um servidor Java EE (JENDROCK et al., 2013).

A JAX-RS converte automaticamente os tipos Java e os MIME do HTTP, como por exemplo, se um método de uma classe Java for marcado com a anotação "*@Produces* (*MediaType.TEXT_PLAIN*)", a API JAX-WS por sua vez converterá o tipo Java para o MIME "*text/plain*" que representa um texto sem formatação e enviará de volta uma reposta ao cliente com o conteúdo dentro do HTTP. Para disponibilizar um recurso para atender as requisições do protocolo HTTP é necessário utilizar as *annotations* (anotações) para identificar o recurso que deseja tornar acessível aos serviços *Web*. A JAX-RS disponibiliza várias *annotations*, dentre as principais podemos citar as seguintes (PENCHIKALA, 2013) e (JENDROCK et al., 2013):

- **@*Path*:** aponta um caminho URI relativo a um determinado recurso. Também sinaliza qual URI será responsável por receber as requisições;
- **@GET:** anotação responsável por definir um acesso de leitura ao recurso;
- **@POST:** tem a função de atualizar um recurso ou criar um novo recurso;
- **@PUT:** responsável por criar um novo recurso;
- **@*DELETE*:** esta anotação possui a função de remover determinado recurso;
- **@*Produces*:** anotação responsável por especificar os tipos de MIME que um recurso pode produzir e enviar de volta para o cliente;
- **@*Consumes*:** determina os tipos de MIME que um recurso poderá receber de um cliente.

Vale lembrar que é possível inserir anotações "*@Produces*" e "*@Consumes*" em uma classe, porém isso implicará que todos os métodos disponíveis nesta classe como recurso produziram ou consumiram, respectivamente de acordo com o tipo definido. Lembrando ainda que se as classes que forem anotadas com "*@Produces"* ou "*@Consumes"* e em seguida o método, a JAX-RS irá dar prioridade a *annotation* mais especifica, que no caso é a *annotation* do método.

4.3. REPRESENTATIONAL STATE TRANSFER (REST)

REST é uma arquitetura de software para desenvolvimento de serviços *Web*, contendo um estilo de distribuir conteúdo de hipermídia, baseando-se na própria estruturada da *Web* (FIELDING, 2000).

O termo REST (*Representational State Transfer*), foi utilizado pela primeira vez por *Roy Thomas Fielding* em sua tese de doutorado. Este estilo de arquitetura pode ser aplicado no desenvolvimento de *Web Services* e também para descrever qualquer interface que tenha que transmitir informações de um domínio específico sobre HTTP, sem fazer uso da camada de mensagens como SOAP ou *session tracking* via *cookies* HTTP. O termo usado para referenciar aplicações que seguem os princípios da arquitetura REST é denominado de "*RESTful*" (MORO; DORNELES; REBONATTO, 2011).

Nesta arquitetura o cliente emite uma requisição ao servidor e este por sua vez responde na forma de representação. Em REST um dos artefatos mais importantes são os recursos, e estes são hospedados em servidores para serem consumidos por clientes. Os recursos são informações como vídeo, imagem, documento entre outros, enquanto a representação nada mais é do que o conteúdo existente em um determinado recurso (SOUZA, 2010).

A Figura 9 ilustra como é o funcionamento de um *Web Services* com os recursos e as representações impostas pela arquitetura REST.

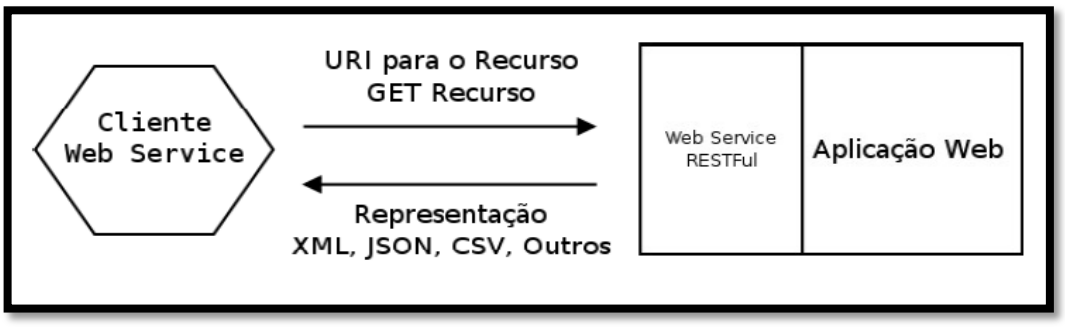

Figura 9 – Web Services REST (In: SOUZA, 2010, p. 15)

Cada recurso é identificado por meio de uma URI, sendo este por sua vez manuseado através do protocolo HTTP, utilizando os verbos comuns a este protocolo como o PUT, GET, POST e *DELETE* (SOUZA, 2010).

O REST é composto por alguns princípios básicos tais como identificação de recurso, onde os recursos são identificados através de URI, à possibilidade manipular os recursos através de suas representações, podendo consultar, criar, modificar ou apagar o recurso tendo a permissão suficiente para efetuar tal operação e por último as mensagens devem ser bem elaboradas contendo as informações suficientes para expor como se processa a mensagem (RICHARDSON; RUBY, 2007).

Devido à facilidade e simplicidade de se desenvolver com o estilo REST, ele vem ganhando força sobre os *Web Services* tradicionais baseado em SOAP, pois estes *Web Services* apresentam uma solução mais voltada para aplicações corporativas de maior grandeza e complexidade, enquanto os baseados em REST para aplicações mais simples voltada para integrar serviços *Web* estratégicos (SOUZA, 2010).

4.3.1. Utilização de protocolos HTTP

O Hypertext Transfer Protocol (HTTP) ou Protocolo de Transferência de Hipertexto é um protocolo de comunicação da camada de aplicação da *Internet,* utilizado para troca ou transferência de hipertexto entre cliente e servidor (FIELDING, 2000).

Para que os componentes de uma arquitetura possam se comunicar é necessário que exista um protocolo. O protocolo HTTP possui uma característica de cliente e servidor, onde a interação entre os componentes é feita através de um modelo de requisição e resposta, ou seja, o cliente emite uma requisição para o servidor apontando para o recurso no qual tem interesse através de uma URI. Esta requisição deve conter informações relativas à ação que o cliente deseja realizar sobre o recurso. O HTTP por ter uma interface uniforme determina uma coleção finita de operações, que podem ser realizadas em qualquer recurso *Web* (POLÔNIA, 2011).

Atualmente, o HTTP é o protocolo mais indicado para trabalhar com *Web Services* REST. O mesmo provê uma interface uniforme e define seis métodos que são empregados na arquitetura orientada a recursos. Os métodos definidos por ROA são: GET, PUT, POST, DELETE, HEAD E OPTIONS, conforme já descritos na seção 2.3.4 deste trabalho (MORO; DORNELES; REBONATTO, 2011).

O HTTP é um protocolo que trabalha sem estado, sendo assim quando um cliente envia uma requisição HTTP, é incluso nela as informações necessárias para que o servidor possa responder a requisição. Sendo assim o servidor não precisa guardar a informação referente à comunicação de seus clientes, ou seja, pode tratar cada requisição como sendo um evento isolado. O protocolo HTTP possui outro aspecto interessante que é o suporte a componentes intermediários na comunicação entre clientes e servidores, permitindo a existência de *proxies* e *gateways*, permitindo o compartilhamento de caches entre clientes distintos, além da possibilidade de encapsular sistemas legados (POLÔNIA, 2011).

Primeiro surgiu a *Internet* e a *World Wide Web*, possibilitando a integração de tecnologias de informação e comunicações, e devido a isso foi possível à criação de um conjunto de especificações e padrões. A ideia de *Web Services* foi criada através de padrões para localização, descrição e convocação de serviços, ou seja, surgiu com o propósito de padronizar e possibilitar a integração de aplicações desenvolvidas em plataformas e sistemas operacionais diferentes.

Nos dias atuais os *Web Services* são utilizados em diversos situações, sendo que na área empresarial, podemos observar uma maior evolução, existindo empresas totalmente dependentes destes serviços, visto que os *Web Services* trazem agilidade e eficiência na comunicação entre aplicações de produção e logística, fazendo que toda e qualquer comunicação entre sistemas passe a ser dinâmica e principalmente segura, pois não há intervenção humana.

4 – TECNOLOGIA GOOGLE ANDROID

O *Android* foi o primeiro projeto de uma plataforma de código aberto voltado a dispositivos móveis lançado no final de 2007 pelo Google em conjunto com um grupo heterogêneo de empresas de TI e telefonia chamada de OHA (*Open Handset Alliance*), tendo como principal objetivo a criação de uma plataforma de desenvolvimento de aplicativos para dispositivos móveis. Esta plataforma é baseada no *Kernel* do sistema operacional *Linux* e diversos aplicativos, com uma interface rica, um navegador de *Internet*, integração com a API do *Google Maps*, suporte à multimídia, GPS, banco de dados *Mobile* entre outras propriedades. O desenvolvimento e interação de aplicações nesta plataforma são codificados através da linguagem de programação *Java* e uma *Integrated Development Enviromnent* (IDE) ou outro ambiente desenvolvimento produtivo e de alto nível (FARTO, 2010).

E com o passar dos anos essa plataforma vem crescendo junto com esse grande avanço tecnológico, onde dispositivos móveis estão ganhando um poder computacional muito grande com relação ao processamento, armazenamento, e comunicação, além de estarem se tornando mais elegantes versáteis, com várias características de *hardware* como GPS, acelerômetro, câmera, entre outros e ao mesmo tempo tornando-se mais acessível aos consumidores (PEREIRA; SILVA, 2009).

Para ter uma noção da popularidade dos dispositivos móveis, hoje existe cerca 6,3 bilhões de celulares em todo o mundo, no entanto a população mundial esta com sete bilhões de habitantes, ou seja, o número de celulares esta quase igualando ao número de habitantes no mundo (TELECO, 2013).

4.1. INTRODUÇÃO

O *Android* é uma plataforma completa para dispositivos móveis, pois envolve um pacote com programas para celulares, já inclusos um sistema operacional, aplicativos, *middleware* e interface de usuário. Esta plataforma foi criada para ser

verdadeiramente de código aberto e com o objetivo de permitir aos desenvolvedores a criação de aplicativos móveis capazes de tirar proveito de toda a capacidade que um aparelho móvel pode oferecer. Um exemplo clássico do que foi dito é a capacidade de uma aplicação interagir com as funcionalidades do núcleo do dispositivo como efetuar chamadas, enviar mensagens de texto, utilizar a câmera, permitindo que o desenvolvedor faça adaptações e evoluam ainda mais estas funcionalidades. Por ser uma plataforma de código aberto esta por sua vez permite que o desenvolvedor faça adaptações, visando incorporar novas tecnologias conforme forem surgindo, visto que existem várias comunidades de desenvolvedores trabalhando em conjunto para construção de aplicações cada vez mais inovadoras (PEREIRA; SILVA, 2009).

As pesquisas feitas atualmente apontam o *Android* como o sistema operacional para *smartphones* que mais cresce no mundo. Este crescimento é embasado justamente pela simplicidade, flexibilidade da arquitetura, e ao mesmo tempo uma ferramenta muito poderosa que permite que seja a base para muitos produtos que se beneficiam dessa plataforma (LECHETA, 2012).

Atualmente a disputa no mercado de mobilidade é muito grande, devido a diversas inovações e lançamentos a todo o momento, fazendo com que até os especialistas sintam dificuldade em acompanhar todo esta evolução. Cada vez mais os dispositivos móveis farão parte do nosso dia a dia, visto que atualmente vivemos a década da mobilidade e o mercado por sua vez, busca cada vez mais por desenvolvedores especializados no assunto para desenvolver aplicações comerciais e corporativas para setores diversos, tais como varejo, economia, saúde, jogos entre outros (LECHETA, 2012).

O mercado corporativo vem acompanhando todo esse crescimento e várias empresas estão buscando incorporar aplicações móveis no seu cotidiano para tornar seus processos mais ágeis e para integrar aplicações móveis com seus sistemas de *back-end*. Devido a forte concorrência que as empresas enfrentam hoje, os celulares e *smartphones* pode ocupar um importante espaço na luta contra a concorrência. As aplicações que executam em um celular hoje podem estar conectadas e online, trocando informações com um servidor confiável da empresa. Atualmente diversos

bancos oferecem serviços a seus clientes, onde é possível acessar a conta corrente e consultar saldo, extratos, e até mesmo pagar contas diretamente de um dispositivo móvel (LECHETA, 2010).

Enquanto as empresas visam uma plataforma moderna ágil e flexível para desenvolver aplicações corporativas para ajuda-las em seus negócios, os usuários comuns visam elegância, modernidade, fácil navegação e uma infinidade de recursos. Em contra partida o *Android* vem suprir as necessidades das empresas, usuários, fabricantes e operadoras de celulares com uma plataforma de desenvolvimento voltada para aplicações móveis baseada em um sistema operacional *Linux*, com várias aplicações já instaladas e com um ambiente de desenvolvimento bastante poderoso e flexível (ABLESON; COLLINS; SEM, 2009) e (FARTO, 2010).

No seu lançamento o *Android* causou um grande impacto, atraindo atenção de muita gente, devido à empresa mentora do projeto ser o *Google* em conjunto com um grupo de empresas chamada de *Open Handset Alliance* (OHA), composta por empresas grandes e conhecidas mundialmente como a Motorola, LG, Samsung, Sony Ericsson e muitas outras (LECHETA, 2010).

4.2. OPEN HANDSET ALLIANCE

A *Open Handset Alliance* (OHA) consiste em um grupo formado por grandes empresas do ramo de telefonia de celulares, companhias de semicondutores, fabricantes de aparelhos celulares, companhias de *software* liderados pelo *Google*. A aliança OHA é composta hoje por aproximadamente 87 empresas, entre elas estão algumas empresas consagradas como a LG, Motorola, Samsung, Sony Ericsson, Toshiba, Sprint Nextel, China *Mobile*, ASUS, Intel, Garmim, ACER, DELL, NVIDIA e muitas outras (OPEN HANDSET ALLIANCE, 2013) e (ABLESON; COLLINS; SEM, 2009).

O objetivo da OHA é definir uma plataforma única e de código aberto para celulares e como isso deixar os usuários mais satisfeitos com o produto final. Outro objetivo

dessa aliança é a criação de uma plataforma moderna e flexível para desenvolvimento de aplicações corporativas. Desta união surgiu o *Android* uma plataforma destinada a desenvolvimento de aplicativos móveis como *smartphones* com um visual moderno, GPS, diversas outras aplicações já instaladas, inovador e flexível e ainda utilizando a já consagrada linguagem de programação Java para desenvolver aplicações usufruindo de todos os recursos que esta linguagem oferece (LECHETA, 2010).

Essa união acaba favorecendo os fabricantes de celulares, os usuários comuns e principalmente os desenvolvedores de aplicações. Os usuários de celulares foram extremamente favorecidos com esta aliança, visto que hoje todos querem dispositivos móveis com um visual moderno e inovador, de fácil usabilidade, com tela *thouch screen*, câmera, jogos, tocador de músicas, GPS, acesso a *Internet* entre outras funcionalidades vão surgindo. E justamente para suprir todo esse anseio dos usuários por vários recursos em um único aparelho que surgiu o *Android* (FARTO, 2010).

A área tecnológica esta sempre evoluindo, e a OHA foi criada como o objetivo de manter uma plataforma padrão que suporte as novas tendências de mercado, englobadas em uma única solução. O fato de o *Android* ser uma plataforma única, de código aberto e possuir uma licença flexível, dando a cada fabricante a oportunidade de realizar as alterações que acharem necessárias, trazem grandes vantagens aos fabricantes de celulares, pois eles podem efetuar alterações no código fonte para customizar seus produtos, sem a necessidade de compartilhar essas alterações com ninguém. Vale lembrar ainda que o fato do *Android* ser de código aberto, também contribui para o seu aperfeiçoamento, visto que existe um grande número de desenvolvedores espalhados em vários lugares do mundo, contribuindo para seu código fonte, adicionando novos recursos e corrigindo falhas (LECHETA, 2010).

Para os desenvolvedores, a plataforma *Google Android* lhe da à oportunidade de usufruir de uma plataforma de desenvolvimento moderna com vários recursos e com tudo o que há de mais moderno.

4.3. SISTEMA OPERACIONAL LINUX

O sistema operacional do *Android* é baseado na versão 2.6 do *kernel* do *Linux* e é responsável por gerenciar os serviços centrais do sistema, tais como gerenciamento de memória, segurança dos arquivos e pastas, processos, threads, pilha de protocolos de rede e modelo de drives. Outra função importante do *kernel* é atuar como uma camada de abstração entre o *hardware* e o restante da pilha de *software* (PEREIRA; SILVA, 2009) e (LECHETA, 2010).

No *Android* cada aplicativo dispara um novo processo no sistema operacional, sendo que alguns podem exibir uma tela para o usuário, enquanto outros podem ficam executando em segundo plano por tempo indefinido. Existe também a possibilidade de diversos processos e aplicativos serem executados simultaneamente, ficando o *kernel* do sistema operacional responsável por controlar os recursos de memória. Caso seja necessário, o sistema operacional pode encerrar algum processo para liberar recursos e memória e depois reiniciar novamente o mesmo processo quando a situação estiver controlada (LECHETA, 2010).

4.4. MÁQUINA VIRTUAL DALVIK

A *Dalvik* é uma máquina virtual preparada para oferecer melhor desempenho, maior integração com novas gerações de *hardware* e projetada para executar várias máquinas virtuais paralelamente. Ela foi projetada para executar em sistemas com pouca memória RAM, baixa frequência de CPU e sistema operacional sem espaço de memória *Swap*, além der ser otimizada para consumir menos memória, bateria e CPU (PEREIRA; SILVA, 2009).

A linguagem Java é utilizada para o desenvolvimento de aplicações para o *Android*, no entanto em seu sistema operacional não existe uma máquina virtual Java (JVM) e sim uma máquina virtual chamada *Dalvik* totalmente preparada para executar em dispositivos móveis (LECHETA, 2010).

Em Java, o código escrito é compilado para gerar os códigos Java usando um compilador Java, para depois estes códigos serem executados pela Java Virtual *Machine* (JVM). No *Android* o processo é diferente. O mesmo código escrito em Java é compilado através do mesmo compilador gerando código Java, mas neste ponto é recompilado novamente usando o compilador *Dalvik* para gerar os códigos *Dalvik*, para em seguida estes códigos serem executados pela máquina virtual *Dalvik* (GARGENTA, 2011).

A Figura 10 faz uma comparação do funcionamento do *standard* Java e do *Android* usando a Dalvik.

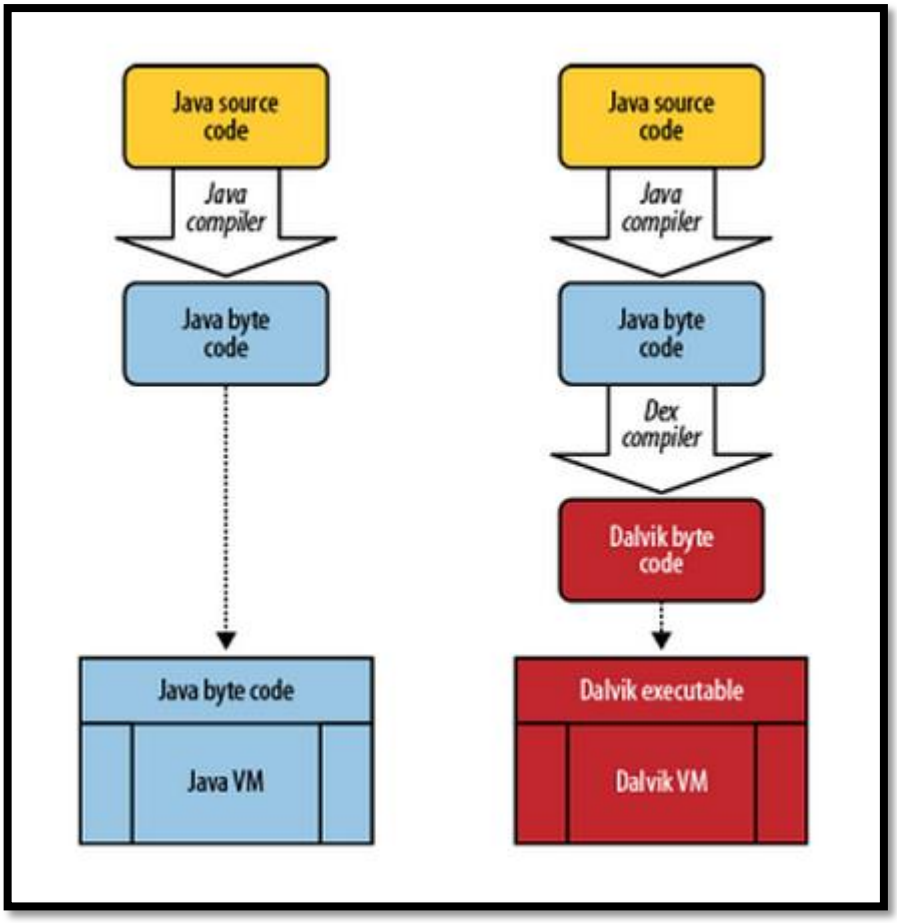

Figura 10 – Java VM versus Dalvik VM (In: GARGENTA, 2011)

No desenvolvimento de aplicações para *Android* as classes Java são compiladas e convertidas para o formato .dex (*Dalvik Executable*), que é a representação de uma aplicação *Android* compilada. Após isso os arquivos .dex e os demais recursos como imagens são compactados em um único arquivo com extensão .apk (*Android Package File*), que representa a aplicação pronta para ser distribuída e instalada (LECHETA, 2010).

4.5. *ANDROID* SDK

O *Android* SDK é o *software* adotado para desenvolver aplicações no *Android*, este por sua vez fornece uma API completa para a linguagem Java, um emulador para simular um celular e ferramentas de desenvolvimento necessárias para construir, testar e depurar aplicativos para o *Android* (FARTO, 2010).

O SDK possui um emulador que pode ser executado como um aplicativo normal, no entanto existe um *plug-in* para o *Eclipse* construído para integrar o ambiente de desenvolvimento Java com o emulador como mostra a Figura 11 (LECHETA, 2010).

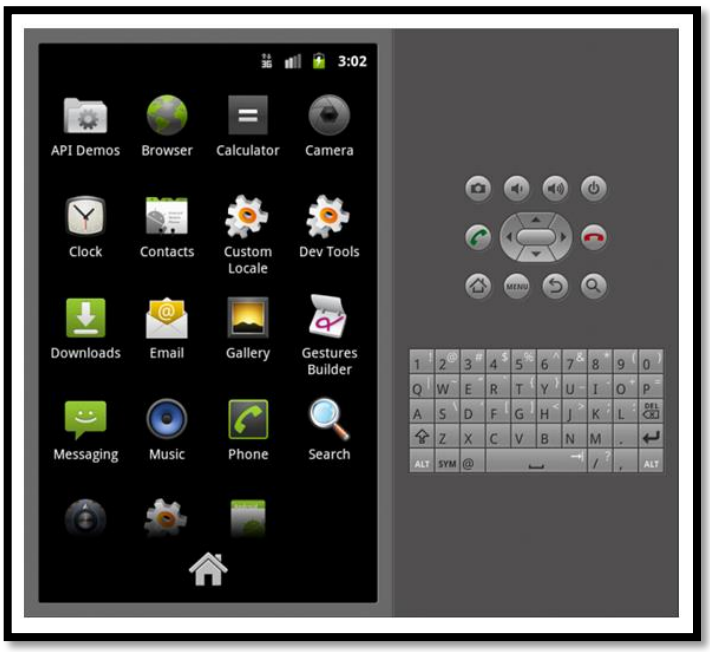

Figura 11 – Emulador do Android Integrado no Eclipse

Com este *plug-in* é possível executar o emulador dentro do *Eclipse*, instalando automaticamente a aplicação no emulador e ainda depurar o código fonte como uma aplicação qualquer da linguagem Java, devido à integração do *debug* do *Eclipse*. Vale lembrar que com este emulador é possível simular aproximadamente 99% dos recursos disponíveis em um dispositivo móvel real. (FARTO, 2010).

4.6. ARQUITETURA *GOOGLE ANDROID*

A plataforma *Google Android* possui uma arquitetura bem estrutura e dividida em cinco camadas. De acordo com a Figura 12 as camadas são divididas em: aplicativos, framework, bibliotecas, ambiente de execução e kernel *Linux* (FARIA, 2008).

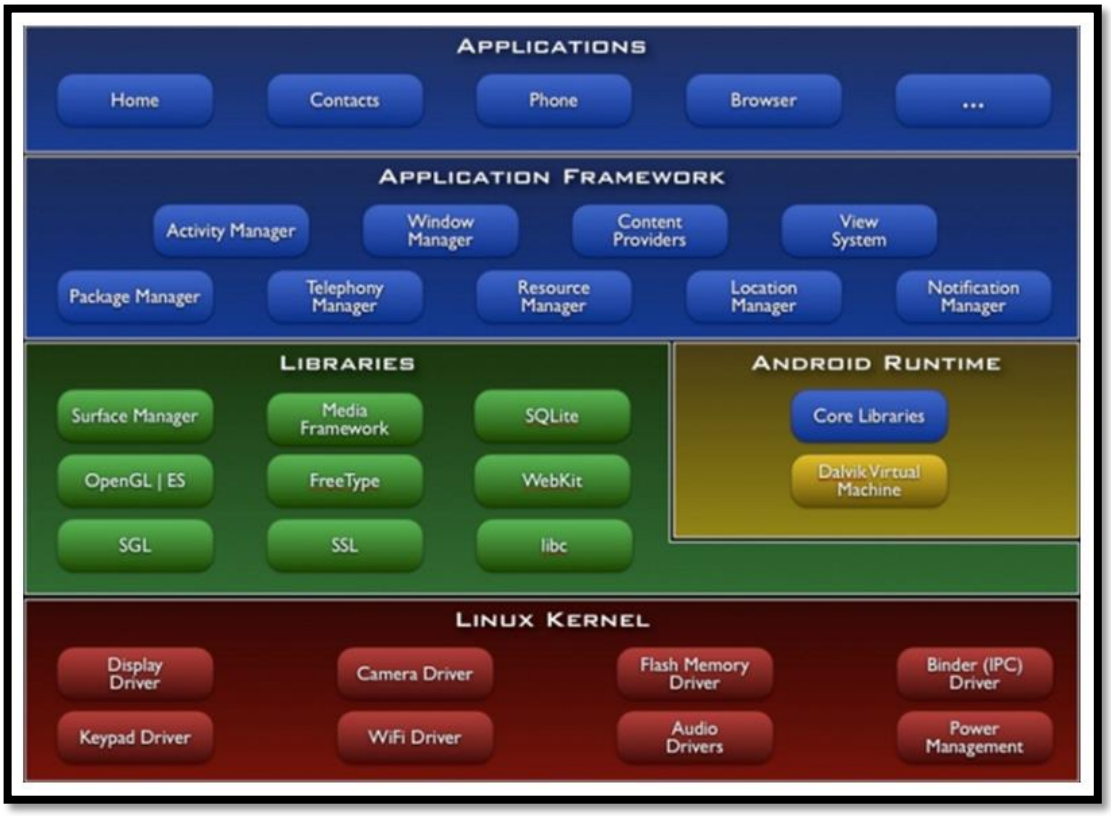

Figura 12 – Camadas da Plataforma Google Android (In: FARIA, 2008)

É fundamental que os desenvolvedores tenham um bom conhecimento do funcionamento dos processos e componentes existentes em cada camada da plataforma, para que possam construir aplicativos mais eficientes e com melhor desempenho (FARTO, 2010).

Nas seções seguintes será feita uma descrição mais detalhada de cada camada, começando pela camada de mais baixo nível, ou seja, de acordo com a Figura 12 de baixo para cima.

4.6.1. Kernel Linux

Nesta camada esta localizada o sistema operacional da plataforma, baseada no *kernel* 2.6 do *Linux*. Ela é responsável pelos serviços de baixo nível da plataforma, tais como segurança, gerenciamento de memória e processos, redes e drives, além de ser responsável pela abstração do *hardware* e o restante do *software*, desempenhando a função de *middleware* entre as camadas da plataforma *Google Android* (FARIA, 2008) e (FARTO, 2010).

Também esta localizada nesta camada um poderoso sistema de gerenciamento de energia, onde através de um aplicativo é feito uma requisição de gerenciamento de energia e o *driver* de energia do *kernel* por sua vez, passa a verificar frequentemente todos os dispositivos que não estão sendo utilizados por aplicações e os desliga (PEREIRA; SILVA, 2009).

4.6.2. Camada *Libraries*

Esta camada contém um conjunto de bibliotecas desenvolvidas em C/C++, utilizadas por vários componentes do sistema. Neste conjunto estão incluídas as bibliotecas C padrão (*Libc*) e também as relacionadas à multimídia, visualização de camadas 2D e 3D, funções para navegadores *Web*, funções para gráficos, funções de aceleração

de *hardware*, renderização 3D, fontes bitmap e vetorizadas e funções de acesso ao banco de dados *SQLite* (FARIA, 2008).

Na Tabela 1 estão descritas as principais bibliotecas da camada *libraries*, assim como uma breve descrição de cada uma delas.

BIBLIOTECA	DESCRIÇÃO
System C Library	Uma implementação derivada da biblioteca C padrão sistema (libc) do BSD sintonizada para dispositivos rodando *Linux*.
Media Libraries	As bibliotecas de mídia suportam os mais populares formatos de áudio, vídeo e imagem.
Surface Manager	Subsistema de exibição, bem como múltiplas camadas de aplicações 2D e 3D
LibWebCore	Um *Web Browser Engine* utilizado tanto no *Android Browser* quanto para exibições *Web*.
Webkit	É um renderizador de páginas para navegadores baseado em código aberto do *Browser Webkit* com suporte para CSS, *javascript*, DOOM e *AJAX*.
SGL	*Engine* de gráficos 2D.
3D libraries	Implementação baseada no *OpenGL*. As bibliotecas empregam aceleração 3D via *hardware* (quando disponível) ou o *software* de renderização 3D altamente otimizado.
FreeType	Renderização de fontes *Bitmap* e Vetor.
SQLite	Um poderoso e leve *Engine* de banco de dados relacional disponível para todas as aplicações *Android*.

Tabela 1 – Principais Bibliotecas da Camada *Libraries* (In: FARTO, 2010)

Conforme descrito na Tabela 1, o *Android* contém várias bibliotecas desenvolvidas em C/C++, e estas por sua vez são utilizadas por diversos recursos do sistema. Estas bibliotecas podem ser acessadas através de *frameworks* disponibilizados para desenvolvedores de aplicativos.

4.6.3. *Android Runtime*

Esta pequena camada é uma instancia da máquina virtual *Dalvik*, criada para cada aplicação executada no *Android*. Devido às limitações dos dispositivos móveis, que possuem pouca memória e velocidade do processador, não foi possível utilizar a máquina virtual do Java, portanto a *Google* tomou a decisão de construir uma máquina virtual que atendesse a essas limitações, foi então que surgiu uma nova máquina virtual chamada *Dalvik*. Também possui várias bibliotecas onde a maioria delas fornecem funcionalidades disponíveis nas principais bibliotecas da linguagem Java como estruturas de dados, acesso aos arquivos, acesso a redes e gráficos (ABLESON; COLLINS; SEN, 2009).

4.6.4. Application Framework

Nesta camada estão localizadas todas as APIs e recursos aproveitados pelos aplicativos, como classes visuais (listas, grades, caixas de texto, botões e navegador *Web*), *View system* (componentes utilizados na construção de aplicativos), provedor de conteúdo (*Content Provider*), que faz com que uma aplicação acesse e até mesmo compartilhe informações de outra aplicação, possibilitando a troca de informações entre aplicativos e gerenciadores de recursos (permite definir e carregar recursos em *Run Time*), gerenciador de localização (GPS e Cell ID), gerenciador de notificação, de pacotes e de atividade, onde é controlado todo o ciclo de vida da aplicação e o acesso e navegação entre as aplicações (FARIA, 2008).

Esta camada também possui outros elementos tais como, o *Location Service*, *Bluetooh Service*, *Wi-Fi Service*, USB *Service*, e Sensor *Service*.

Na Tabela 2 estão os principais elementos localizados nesta camada assim como uma breve descrição de cada um dos recursos.

RECURSO	DESCRIÇÃO
Activity Manager	Faz o gerenciamento do ciclo de vida de todas as *activities*, ou seja, quando iniciar e terminar, permitindo o deslocamento de uma *activity* para outra.
Package Manager	A *activity manager* utiliza este elemento para ler as informações dos pacotes do arquivo *Android* (APK). O *package manager* faz a comunicação com o restante do sistema informando quais pacotes estão sendo utilizados no dispositivo e quais as capacidades destes pacotes.
Windows Manager	Basicamente faz o gerenciamento das apresentações de janelas, informando qual janela estará ativa.
Content Providers	Possibilita o compartilhamento de dados entre os aparelhos, possibilitando a troca de informações entre os aplicativos.
View System	Disponibiliza todo o tratamento gráfico para a aplicação, como botões, layouts e frames.

Tabela 2 – Principais Recursos da *Framework* (In: PEREIRA; SILVA, 2009)

Na plataforma *Android* todos os conjuntos disponíveis possuem total acesso às APIs utilizadas pelo núcleo da plataforma. Esta arquitetura foi projetada de forma a simplificar o reuso de componentes, fazendo com que uma aplicação possa publicar as suas capacidades e outra aplicação qualquer então, pode fazer uso dessas capacidades (PEREIRA; SILVA, 2009).

4.6.5. Application

Nesta camada encontramos os aplicativos fundamentais escritos em Java como, mapas, navegadores, calendários, programas de SMS, gerenciador de contatos, cliente de e-mail, agendas entre outros (FARIA, 2008).

Nesta camada estão todos os aplicativos desenvolvidos para a plataforma *Google Android*. Eles podem vir previamente instalados no dispositivo ou desenvolvidos por terceiros, ou seja, para o *Android* não existe diferença entre aplicações nativas e aplicações desenvolvidas por outras pessoas, isso garante a plataforma um alto grau de flexibilidade e extensibilidade (GARGENTA, 2011) e (TOSIN, 2011).

Para desenvolver aplicações para a plataforma *Android* basta escrever os códigos em Java para depois serem executados na máquina virtual *Dalvik*. Vale lembrar ainda que o Java possui uma grande portabilidade, sendo possível incorporar outras aplicações escritas em Java com o *Android*. (PEREIRA; SILVA, 2009).

4.7. COMPONENTES DE UMA APLICAÇÃO *ANDROID*

A plataforma *Android* possui um grupo de componente a disposição dos desenvolvedores para que o objetivo final da aplicação seja atingido. Estes componentes são essenciais para que o sistema possa instanciar e executar sempre que seja necessário. A Figura 13 ilustra cada um destes componentes (TOSIN, 2011) e (PEREIRA; SILVA, 2009).

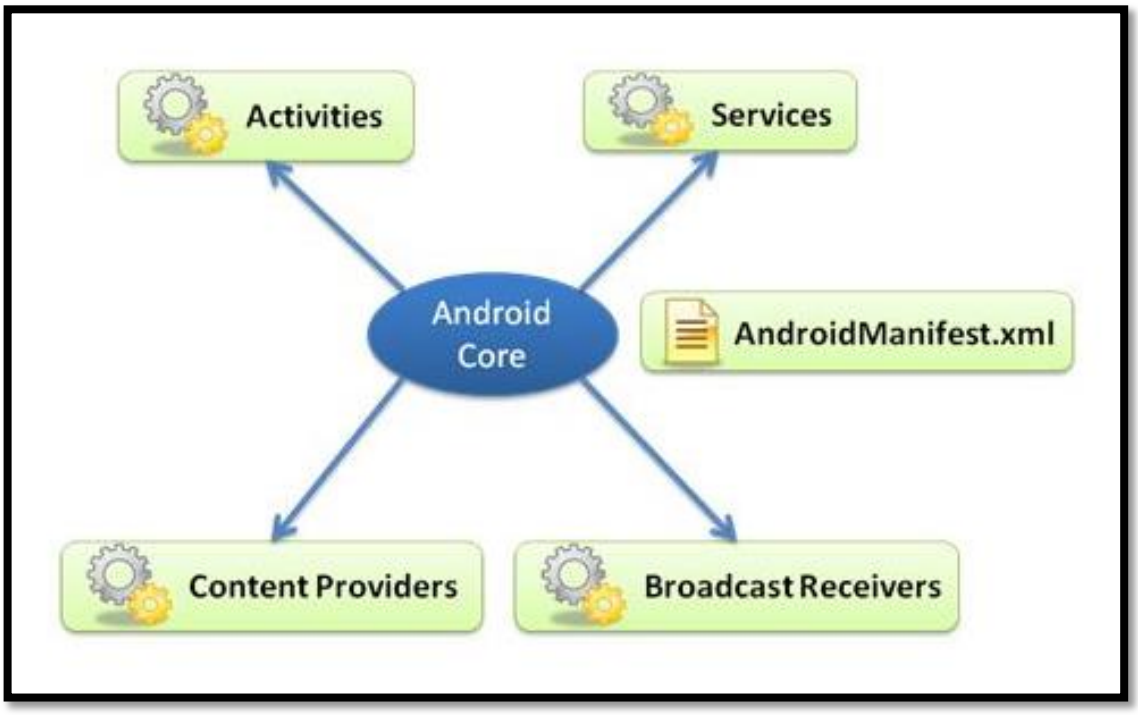

Figura 13 – Componentes de Uma Aplicação *Android* (In: TOSIN, 2011)

Como mostrado na Figura 13 os componentes são: *Activities*, *Services*, *Broadcast Receivers*, *Content Providers* e junto com estes componentes também temos o *AndroidManifest.xml*. A seguir será descrito mais detalhadamente cada um destes componentes:

- **Activity:** este componente é o mais utilizado, visto que cada *activities* esta associada a uma *view*, onde são definidas como será feita a exibição visual para o usuário. As *activities* tem a responsabilidade de gerenciar os eventos de tela e coordenar o fluxo da aplicação, ou seja, cada *activity* exibe uma interface gráfica e responde a eventos iniciados pelo sistema e pelo usuário (TOSIN, 2011).
- **Services:** os *services* são códigos que executam em segundo plano e geralmente são utilizados para tarefas que requerem um grande tempo de execução. Estes códigos não possuem interfaces de usuário, rodam em *background* e não são interrompidos quando é feita a troca de atividade pelo usuário. Os *services* permanecem ativos até que recebe outra ordem. Depois de estabelecido conexão com o serviço, é possível efetuar comunicação através de uma interface apresentada pelo usuário (PEREIRA; SILVA, 2009).
- ***Content Providers*:** os *content providers* ou provedores de conteúdo é basicamente uma forma de compartilhar dados entre as aplicações que executam no dispositivo. Eles permitem que outras aplicações tenham a possibilidade de armazenar e recuperar dados no mesmo repositório utilizado por outra aplicação. Um exemplo disso é uma aplicação de gerenciamento de contatos do *Android* nativa, onde aplicações desenvolvidas por terceiros podem utilizar um *content provider* com o intuito de ler os contatos armazenados no dispositivo de forma simples (TOSIN, 2011).
- ***Broadcast Receivers*:** Os *broadcasts receivers* são componentes que ficam “ouvindo” a ocorrência de determinados eventos, sendo que estes podem ser nativos ou disparados por aplicações. Por exemplo, podemos ter uma aplicação utilizando um *broadcast receiver* para avisar quando o dispositivo estiver recebendo uma ligação e com base nessa informação efetuar algum tipo de processamento. Este componente não possui interface de usuário,

> mas existe a possibilidade de fazer uso de uma *Notification Manager* como forma de interface para notificar o usuário sobre alguma ocorrência. Vale lembrar que não é necessário a aplicação estar ativa para que o *broadcast receiver* seja acionado (PEREIRA; SILVA, 2009).

Junto a esses componentes existe um arquivo de manifesto chamado "*AndroidManifest.xml*". Cada aplicação deve ter um arquivo deste em seu diretório raiz, visto que é obrigatório e único para cada aplicação. As configurações gerais da aplicação, assim como os componentes que fazem parte da mesma são feitas no manifesto *Android* (TOSIN, 2011).

Este arquivo permite fazer a configuração de todos os elementos da aplicação, tais como as classes de cada componente a ser utilizado, qual o tipo de dado ele pode tratar, quando pode ser ativado, ou seja, define os dados de cada elemento presente na aplicação. Também deve conter neste arquivo todas as especificações de permissões necessárias para a aplicação, como por exemplo, a permissão de acesso à rede (PEREIRA; SILVA, 2009).

Podemos observar que a plataforma *Google Android* foi bem aceita no mercado mundial e continua crescendo muito nos últimos anos. O fato de ser *Open Source*, possuir uma arquitetura bastante simples e flexível, ser liderada pela *Google* juntamente com outras empresas de grande porte, principalmente da área de telefonia, contribuiu para essa grande aceitação por parte dos usuários e desenvolvedores.

Embora seja um sistema operacional para dispositivos móveis onde os recursos de memória e *hardware* é bem restrito, esta plataforma é bastante robusta e oferece várias ferramentas e recursos tanto aos usuários como aos desenvolvedores. Esta plataforma apresenta uma arquitetura muito bem estruturada, com diversas bibliotecas e elementos que contribuem para o desenvolvimento de aplicações robustas e inovadoras.

5 – PROPOSTA DO TRABALHO

A proposta deste trabalho é construir uma aplicação *Web*, capaz de expor seus serviços através de *Web Services*, para que uma aplicação *Mobile* possa consumir estes serviços. A aplicação móvel é responsável por colher às informações geradas de forma externa ao ambiente de trabalho, salva-las em seu banco de dados para posteriormente serem enviadas via *Internet* para a aplicação *Web*, garantindo assim uma maior confiabilidade e integridade das informações.

Além disso, a aplicação *Mobile* possui um processo de sincronização com a *Web,* onde é possível sincronizar os dados necessários para efetuar a coleta de dados externos, garantindo mais confiabilidade nas informações geradas. Portanto este projeto tem como propósito geral, melhorar processo de captura das informações geradas externamente, facilitar o trabalho dos funcionários e melhorar a qualidade das informações geradas, tendo um controle mais eficaz das informações geradas.

Nas seções seguintes serão abordados os conceitos que foram utilizados para construção da aplicação *Web*, a *Mobile,* os *Web Services*, as tecnologias envolvidas, a arquitetura e os modelos utilizados no processo de construção do projeto.

5.1. AMBIENTE *WEB*

Para a construção do ambiente *Web* foi utilizado à linguagem de programação Java e a ferramenta Java EE, visto que contém vários recursos para facilitar a construção do ambiente *Web*.

Dentre os recursos existentes nesta plataforma, foi abordado na construção deste ambiente Web, o *Java Persistence* (JPA) que faz a persistência de dados, o *Java API for XML Binding* (JAXB) para trabalhar com arquivos XML e *Web Services*, também foi utilizado o *Jersey* que contém basicamente um servidor e um cliente REST, e por último o servidor de aplicações Java para *Web* chamado *Tomcat*. Também foi utilizado o *framework JavaServer Faces* (JSF) que estabelece um

padrão para a construção de interfaces com o usuário do lado do servidor e para melhorar a interface da aplicação foi utilizada a biblioteca de componentes *PrimeFaces*.

Para armazenar as informações geradas pelas aplicações foi utilizado o banco de dados *MySQL*. Este banco possui código aberto, é confiável, fácil de usar, além de ser muito utilizados em aplicações *Web*.

De uma maneira geral a aplicação *Web* disponibilizará interfaces para o usuário efetuar as movimentações necessárias da aplicação, além de disponibilizar os recursos necessários para serem consumidos pela aplicação *Mobile*, ou seja, *Web Services* baseado no modelo de arquitetura RESTFUL.

5.2. WEB SERVICES RESTFUL

Nos dias atuais as empresas precisam enviar as informações de seu cotidiano de forma rápida, segura e eficiente, devido à necessidade de ter os dados disponíveis para serem tomadas as devidas ações em caso necessário, seja para melhorar suas vendas ou para melhorar algum processo.

Os *Web Services* vem para contribuir na transmissão dos dados gerados fora da empresa, tornando o processo de inserção dos dados no sistema de forma rápida eficiente e com menor custo, garantindo assim que as informações estejam disponíveis para as tomadas de decisões.

Na aplicação desenvolvida foi implementado *Web Services*, usando as técnicas dos padrões *Representational State Transfer (*REST). Estes padrões utilizam o protocolo HTTP que é bastante rico para construção de *Web Services*, porém apresenta uma forma mais simplificada (RICHARDSON; RUBY, 2007).

Também foi utilizado a Arquitetura Orientada a Recursos (ROA) que promove um modelo de boas práticas, onde é possível unificar os princípios de REST com as tecnologias da *Web*. Com REST e ROA juntos, podemos desenvolver aplicações com baixo acoplamento, com melhor adaptabilidade e interoperabilidade (FRANÇA, et al., 2011).

5.3. PROJETO *MOBILE* COM *ANDROID*

Neste trabalho foi construída uma aplicação *Mobile* utilizando a plataforma *Google Android*. Como já foi descrito anteriormente, esta plataforma esta crescendo muito nos últimos anos e atualmente detém mais de 81,3% do mercado mundial. Devido a essa proporção mundial que foi escolhida esta plataforma para construção da aplicação móvel.

A aplicação *Mobile* é responsável por colher às informações geradas fora do ambiente interno das empresas e ao chegar a um local com boa conexão com a *Internet* o usuário poderá enviar os dados direto para a aplicação *Web*, além de possibilitar a sincronização de informações necessárias, tais como cadastros de clientes, funcionários, produtos entre outros que auxiliará na coleta de dados fora da empresa. A Figura 14 ilustra como é o processo de integração entre as aplicações e os bancos de dados.

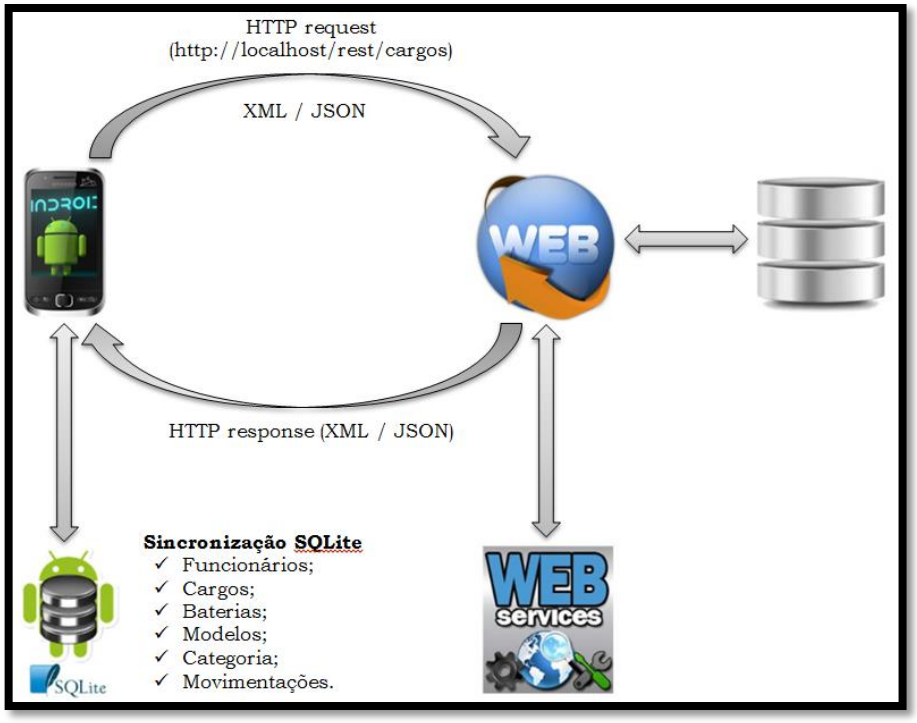

Figura 14 – Processo de Integração entre as Aplicações e os Bancos

Como mostrado na Figura 14, as movimentações efetuadas no campo ficam armazenadas no banco *SQLite* do dispositivo móvel, para posteriormente serem enviadas para a aplicação *Web*. Isso se faz necessário, devido vários lugares no campo não ter cobertura de *Internet*, principalmente em lugares com baixa altitude. Também é possível perceber que o dispositivo móvel tem um mecanismo de sincronização com a aplicação *Web* para atualizar as informações necessárias para efetuar as movimentações tais como funcionários, cargos, categoria operacional, modelos e baterias. Vale lembrar que as informações da aplicação móvel interagem com a aplicação *Web* através de requisições HTTP, utilizando o formato de JSON ou XML.

Neste capítulo foi descrito como foi construído as aplicações, ou seja, como foi construída a parte da *Web*, a integração com a aplicação *Mobile* através de *Web Services*, utilizando os padrões de REST e o modelo de ROA entre outras tecnologias envolvidas, assim como os padrões de projeto a serem utilizados. De uma maneira geral foi explicado como foi construído o projeto, usando boa parte do material e vivenciando na prática todo conteúdo pesquisado neste trabalho.

6 – ESTUDO DE CASO

Para expor o todo o material pesquisado e validar a proposta deste trabalho através de um problema real, foi abordado o desenvolvimento de uma solução para um problema existente hoje nas trocas de baterias dos equipamentos em usinas de açúcar e álcool.

Atualmente esse processo é feito através de apontamento manual e posteriormente digitado no sistema da empresa, ou seja, o eletricista vai para o campo fazer a substituição da bateria e neste momento faz o apontamento para posteriormente ser inserido no sistema e em muitos casos nem isso é feito, ficando sem controle algum. Este processo acaba sendo muito lento e gerando muita perda de informação devido ao não preenchimento correto das informações necessárias, caligrafia ilegíveis, esquecimento na entrega dos apontamentos, números das baterias e equipamentos incorretos, o funcionário acaba deixando para fazer o apontamento no dia seguinte e não lembra mais das informações necessárias, além da empresa ter que disponibilizar um funcionário para digitar os apontamentos que chegam do campo.

Como visto anterior, foi apresentado à estrutura geral do projeto, abordando as tecnologias os conceitos e a arquitetura que foram utilizadas para o desenvolvimento do projeto para auxiliar no controle de baterias. Neste capítulo será apresentado como foi construído esse projeto, explicando detalhes do desenvolvimento da aplicação *Web*, *Mobile* e os *Web Services*.

6.1. DESENVOLVIMENTO DO AMBIENTE *WEB*

O ambiente *Web* foi desenvolvido na linguagem Java, utilizando a IDE Java EE e o servidor da *Apache Tomcat*. O banco de dados *MySQL*, além de outras tecnologias serão comentadas no decorrer do texto. A Figura 15 apresenta o nome do projeto e os pacotes existentes na estrutura da aplicação *Web*.

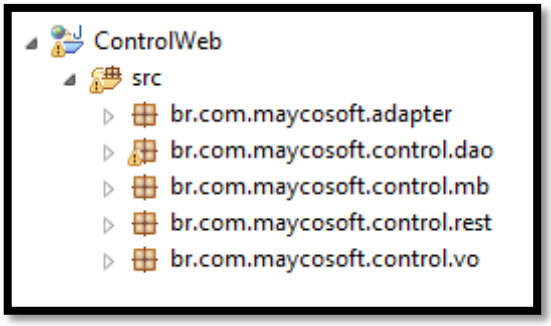

Figura 15 – Pacotes da Aplicação *Web*

Como mostrado na Figura 15 o projeto esta dividido em cinco pacotes visando uma melhor organização dos códigos no desenvolvimento do projeto. O primeiro pacote chamado de "*br.com.maycosoft.adapter*" é bem simples, possui somente uma classe com o nome de "*DateAdapter*", responsável pela manipulação de data na aplicação e o pacote "*br.com.maycosoft.control.rest*" será explicado posteriormente.

Nas seções seguintes serão descritos o conteúdo dos pacotes *Value Object (VO)*, *Acess Object (DAO)* e *ManagedBean (MB)* na sequencia que foram criados para um melhor entendimento, haja visto que uma classe depende da outra.

6.1.1. Pacote *Value Object* (VO)

Neste pacote estão às classes que contém as variáveis, os construtores e os *getters* e *setters*. Outro fato importante neste pacote são as anotações existentes nas classes para mapear os atributos da classe para o banco de dados. A JPA é responsável pela persistência dos dados, permitindo que o desenvolvedor possa efetuar o mapeamento, para em seguida armazenar, atualizar e recuperar os dados de bancos de dados relacionais para objetos Java e vice-versa, permitindo que o programador trabalhe diretamente com objetos ao invés de instruções SQL.

Todas as classes que deverão ser mantidas em um banco de dados devem ser anotadas, ou seja, ao anotar uma classe com a expressão "*@Entity",* a JPA se encarregará de criar uma tabela para a entidade no banco de dados da aplicação.

Por padrão, o nome da tabela será o mesmo da classe, porém através da anotação "*@Table* (*name* = "*NomeDaTabela*")" é possível alterar. Também é possível alterar o nome padrão da coluna via "*@Column* (*name* = "*NomeDaColuna*")".

Lembrando ainda que podemos indicar qual será a chave primária anotando "*@Id*" antes do atributo desejado e se o desenvolvedor preferir poderá acrescentar à expressão *"@GeneratedValue(strategy=GenerationType.IDENTITY)*" para habilitar o autoincremento na tabela.

Em seguida é apresentado um exemplo de como foi implementada a classe "CargoVO" da aplicação para melhor entendimento do que foi descrito. Devido à aplicação fazer uso de XML é necessário fazer algumas anotações para a *Java Architecture for XML Binding* (JAXB) possa ler e escrever os objetos *Java* para documentos XML. A seguir é mostrada uma parte do código onde podemos ver as seguintes anotações: *"@XmlRootElement*", *"@XmlType(propOrder* = { "codigo", "descricao" })" e a *@XmlAccessorType(XmlAccessType.FIELD)*", responsáveis pela manipulação de arquivos XML.

```
package br.com.maycosoft.control.vo;

import java.io.Serializable;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.Table;
import javax.xml.bind.annotation.XmlAccessType;
import javax.xml.bind.annotation.XmlAccessorType;
import javax.xml.bind.annotation.XmlRootElement;
import javax.xml.bind.annotation.XmlType;

import org.codehaus.jackson.annotate.JsonProperty;

@Entity
@Table(name = "MODELOS")
@XmlRootElement
@XmlAccessorType(XmlAccessType.FIELD)
@XmlType(propOrder = { "codigo", "descricao" })
public class ModeloVO implements Serializable {
```

```
    private static final long serialVersionUID = 2642108556749509783L;

    @Id
    @GeneratedValue(strategy=GenerationType.IDENTITY)
    @Column(name = "CODIGO")
    private int codigo;

    @Column(name = "DESCRICAO", length = 100, unique = true)
    private String descricao;
```

Por padrão os campos são mapeados para a coluna como o nome dos campos, mas é possível alterar o nome padrão via "*@Column* (*name* = "*NomeDaColuna*)".

De uma maneira geral as classes do pacote "VO" são responsáveis por mapear a classe para o banco de dados conforme ilustra a Figura 16. A JPA faz o mapeamento dos atributos da classe criando uma tabela com os atributos da classe conforme as anotações feitas pelo desenvolvedor.

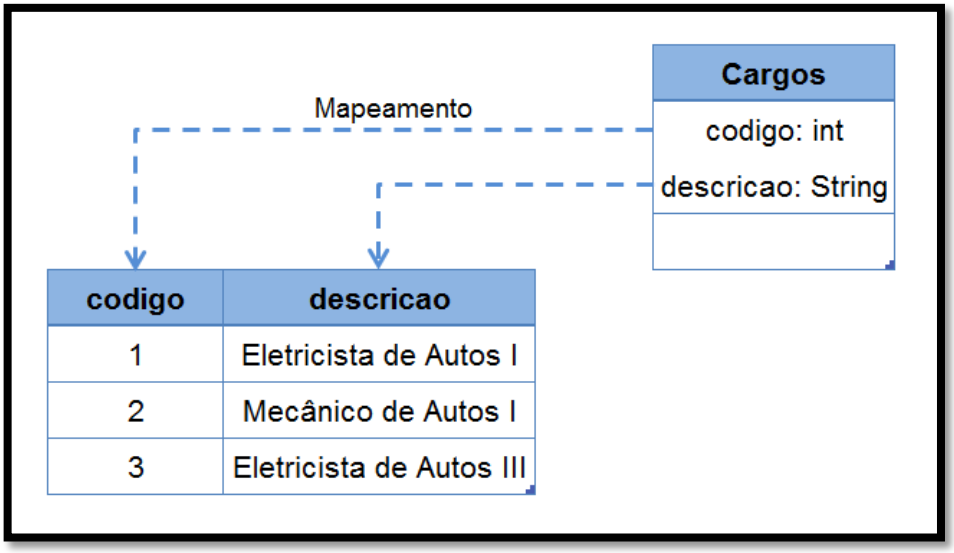

Figura 16 – Exemplo de Mapeamento de uma Classe para o Banco

A Figura 16 demonstra um exemplo de como é feito o mapeamento da classe "Cargos" com seus respectivos atributos, para uma tabela do banco de dados relacional.

6.1.2. Pacote *Data Acess Object* (DAO)

No pacote DAO estão às classes responsáveis por toda transação realizada no banco de dados, tais como, criação de transação com o banco, *comit*, *rollback* e encerramento da conexão com o banco de dados entre outras.

Para um melhor entendimento das classes DAO, antes é necessário apresentar o arquivo "*persistence.xml*", criado na pasta "*META-INF*" de cada projeto, seja ele *Web* ou *Desktop* conforme ilustrado na Figura 17.

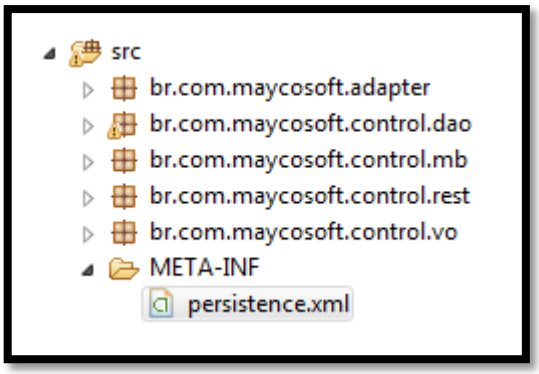

Figura 17 – Localização do Arquivo *Persistence.xml*

Este arquivo armazena as informações de configuração referentes ao provedor da JPA e o banco de dados, além de podermos identificar as classes que serão mapeadas como entidades do banco de dados relacional. A seguir é apresentado o conteúdo deste arquivo desenvolvido neste projeto.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0" xmlns="http://java.sun.com/xml/ns/persistence"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd">
      <persistence-unit name="ControlWebPU" transaction-type="RESOURCE_LOCAL">
          <provider>org.eclipse.persistence.jpa.PersistenceProvider</provider>
```

```
            <class>br.com.maycosoft.control.vo.FuncionarioVO</class>
            <class>br.com.maycosoft.control.vo.ModeloVO</class>
            <class>br.com.maycosoft.control.vo.CategoriaVO</class>
            <class>br.com.maycosoft.control.vo.BateriaVO</class>
            <class>br.com.maycosoft.control.vo.CargoVO</class>
            <class>br.com.maycosoft.control.vo.EquipamentoVO</class>
            <class>br.com.maycosoft.control.vo.MovBateriaVO</class>

            <properties>
              <property name="javax.persistence.jdbc.url"
                        value="jdbc:mysql://localhost:3306/mayco"/>
               <property name="javax.persistence.jdbc.driver"
                         value="com.mysql.jdbc.Driver"/>
               <property name="javax.persistence.jdbc.user"

                                 value="root"/>
                        <property name="javax.persistence.jdbc.password"
                                  value="semp"/>
                        <property name="eclipselink.ddl-generation"
                                  value="create-tables"/>
          </properties>

        </persistence-unit>

</persistence>
```

Como podemos observar, no código apresentado foram identificadas quais as classes que serão mapeadas como entidades do banco de dados e as informações de conexão do banco, como URL, *driver* JDBC, usuário e senha do banco. Observa-se também que na *tag* marcada em amarelo, é definido através da propriedade "*name"* o nome que será utilizado, quando criado um *EntityManager* de um *EntityManagerFactory*. Invocando o método apresentado no trecho de código a seguir, é possível recuperar uma fábrica de *EntityManager* a partir do arquivo "*persistence.xml"*.

```
package br.com.maycosoft.control.dao;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.EntityTransaction;
import javax.persistence.Persistence;
import javax.persistence.Query;
import br.com.maycosoft.control.vo.CargoVO;
```

```
public class CargoDAO {
    private static final EntityManagerFactory factory = Persistence
                .createEntityManagerFactory("ControlWebPU");
    private EntityManager em = null;
    private EntityTransaction transaction = null;

    public void beginTransaction() {
          em = factory.createEntityManager();
          transaction = em.getTransaction();
          transaction.begin();
    }
```

Recuperando a fábrica de *EntityManagerFactory*, é possível criar consultas, buscar objetos, sincronizar objetos e inserir objetos no banco de dados, visto que a *EntityManager*, provê vários recursos para efetuar estas transações. A seguir é apresentado um fragmento de código, mostrando como foi construído o método para realizar a inserção de dados no banco.

```
   public void insert(CargoVO cargo) {

          try {
                this.beginTransaction();
                em.persist(cargo);
                this.commit();

          } catch (Exception e) {
                this.rollback();

          } finally {
                this.closeConnection();
          }

    }
```

Como podemos observar o método para inserção utiliza poucas linhas de código, pois inicia a transação no banco, faz a persistência dos dados, efetua o *commit*, se der algum erro faz o *rollback* e por fim fecha a conexão com o banco de dados. Isso por que a JPA possibilita que o desenvolvedor trabalhe diretamente com objetos ao invés de instruções SQL.

A JPA é uma especificação que atualmente possui várias implementações disponíveis, dentre elas o *Hibernate*, *TopLink*, *Kodo*, Apache *OpenJPA* e o *EclipseLink* que foi usado para implementação desse projeto. O *EclipseLink* é um projeto de código aberto, que possibilita ao desenvolvedor *Java* interagir com diversos tipos de serviços tais como objetos XML, banco de dados, *Web Services* e muitos outros.

6.1.3. Pacote *ManagedBean* (MB)

As classes deste pacote são classes normais de Java, contêm os métodos *getters* e *setters*, regras de negócios, porém mapeadas através de anotações ou *Java Annotations*. São classes *Java Bean* regularmente registrada com o JSF, ou seja, é um *Java Bean* gerenciado pelo *framework* JSF. Sendo assim o *Managed Bean* pode ser acessado a partir da página JSF. A seguir é apresentado um fragmento de código exemplificando o método “salvar” da classe *Managed Bean* e suas anotações.

```
package br.com.maycosoft.control.mb;

import java.util.List;

import javax.faces.bean.ManagedBean;
import javax.faces.bean.ViewScoped;

import br.com.maycosoft.control.dao.CargoDAO;
import br.com.maycosoft.control.vo.CargoVO;

@ManagedBean(name = "cargoMB")
@ViewScoped
public class CargoMB {

   private CargoVO cargo = null;
   private List<CargoVO> cargos = null;
   private CargoDAO dao = null;
   private boolean isAlterar;

   public CargoMB() {

         cargo = new CargoVO();
```

```
            dao = new CargoDAO();
            cargos = dao.selectAll();
            isAlterar = false;
      }
      // Omitidos os getters e setters

      public void salvar() {

            if (!isAlterar) {
                  dao.insert(cargo);
            } else {
                  dao.update(cargo);
            }

            cargos = dao.selectAll();
            cargo = new CargoVO();
            isAlterar = false;

      }
```

No código mostrado é possível observar a anotação "*@ManagedBean* (*name* = "*cargoMB*")", responsável por especificar o nome do atributo, se não for atribuído um nome, por padrão será o nome da classe. Já a anotação "*@ViewScoped*" é um *bean* que fica ativo enquanto o usuário esta interagindo com a mesma página JSF na janela do *browser*.

6.1.4. Recursos Utilizados no Projeto

Para o desenvolvimento deste projeto foi necessário importar várias bibliotecas para a IDE *Eclipse* dentre as principais podemos destacar o *EclipseLink*, *Jersey*, *Mojarra* e *PrimeFaces*. A seguir será descrito um comentário para cada item.

- ***EclipseLink:*** é um projeto que implementa serviços suportados por uma série de normas de persistência de dados incluindo a JPA, a JAXB entre outras. Ele fornece várias soluções aos desenvolvedores, baseadas em padrões de persistência objeto-relacionais, com suporte adicional para diversas características avançadas (VOHRA, 2010).
- ***Jersey:*** contém basicamente um servidor REST e um cliente REST, sendo que do lado do cliente fornece uma biblioteca para se comunicar com o

servidor. Do lado do servidor ele usa um *servlet* que verifica as classes para identificar os recursos *RESTful*. Através do arquivo "*web.xml"* é possível registrar o *servlet* que será fornecido pelo *Jersey* (BURKE, 2010).

- ***Mojarra:*** é uma especificação de implementação para JSF, ela define o modelo de desenvolvimento e oferece alguns componentes bem básicos, como *inputs*, botões e *comboBox*.
- ***PrimeFaces:*** é um *framework open-source* que oferece diversos componentes para *JavaServer Faces* (JSF), e permite a aplicação de temas, com o objetivo de melhorar a aparência dos componentes de forma muito simples.

6.1.5. Páginas *Web* da Aplicação

As páginas *Web* são as interfaces de interação com o usuário. Neste projeto como já dito anteriormente as páginas foram criadas através de uma implementação JSF, onde as páginas *Web* possuem extensão "**.xhtml*". Estes arquivos ficam armazenados na pasta "*WebContent"*, como mostra a Figura 18 abaixo.

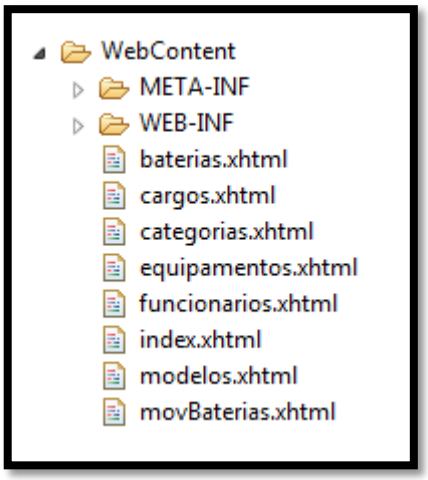

Figura 18 – Arquivos XHTML

Na Figura 18 é possível observar vários arquivos com extensão “.xhtml”, sendo que cada arquivo representa uma página da aplicação *Web*.

O *Extensible Hypertext Markup Language* ou XHTML é uma reformulação da linguagem de marcação HTML baseada em XML, combinando *tags* de marcação HTML com as regras de XML, melhorando a acessibilidade, podendo ser interpretado por qualquer dispositivo, independentemente da plataforma utilizada.

A seguir é apresentado um fragmento de código da tela de cargos da aplicação *Web* para ficar mais claro o entendimento do XHTML.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
    "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml"
      xmlns:ui="http://java.sun.com/jsf/facelets"
      xmlns:h="http://java.sun.com/jsf/html"
      xmlns:f="http://java.sun.com/jsf/core"
      xmlns:p="http://primefaces.org/ui">

<h:head>
      <title>Cargos</title>
</h:head>
<h:body>

      <h:form id="formCargos">
            <h:messages />
            <p:panel id="cargosPanel" header="Cadastro de Cargos">
                  <h:panelGrid columns="6">
                        <h:outputLabel for="txtCodigo" value="Codigo" />
                        <p:inputText id="txtCodigo" disabled="true"
value="#{cargoMB.cargo.codigo}"
                              size="2" />
                        <h:outputLabel for="txtDescricao" value="Descricao" />
                        <p:inputText id="txtDescricao"
value="#{cargoMB.cargo.descricao}"
                              size="40" />
                        <h:outputLabel for="txtSigla" value="Sigla" />
                        <p:inputText id="txtSigla"
value="#{cargoMB.cargo.sigla}" size="3" />
                  </h:panelGrid>

            </p:panel>

     Foram omitidos os demais códigos
```

Através das *tags* apresentada anteriormente no trecho de código é possível formar a interface gráfica ilustrada na Figura 19. Esta página oferece ao usuário a opção para cadastrar cargos, as demais telas do ambiente *Web* seguem o mesmo padrão. A aplicação ainda disponibiliza páginas para cadastro de equipamentos, baterias, categorias operacionais, funcionários, modelo e página para efetuar as movimentações de baterias.

Figura 19 – Página de Cadastro de Cargos

A aparencia da tela apresentada na Figura 19 é uma de muitas outras existentes. O *PrimeFaces* possui vários outros temas disponíveis no site. Para mudar a aparencia da página, é preciso baixar o tema escolhido, salvar na pasta "*lib*" e efetuar a configuração do arquivo "*web.xml*", informando o nome do novo tema.

Nesta seção foi apresentado a forma que foi construido a aplicação *Web*, mostrando a estrutura do projeto, as tecnologias utilizadas, a arquitetura e as bibliotecas incorporadas ao projeto para melhorar e facilitar o desenvolvimento da aplicação *Web*. Dessa forma foi possível vivênciar na prática o material pesquisado garantindo assim um melhor entendimento das tecnologias envolvidas no desenvolvimento do ambiente *Web*.

6.2. DESENVOLVIMENTO DOS *WEB SERVICES RESTFUL*

A implementação dos *Web Services* neste projeto irá oferecer mecanismos para a aplicação móvel desenvolvida na plataforma *Android*. As movimentações de baterias ficam armazenadas no banco de dados da aplicação *Mobile* e transmitidas para a aplicação *Web*, quando o usuário estiver em local com boa cobertura de *Internet*.

Os *Web Services* da aplicação estão localizados no pacote "*br.com.maycosoft.control.rest*" como mostrado na Figura 20. Dentro do pacote estão todas as classes anotadas para oferecer serviços para a aplicação móvel. Nestas classes foram utilizados as anotações da API "JAX-RS", que simplifica o desenvolvimento de aplicações que fazem uso da arquitetura REST.

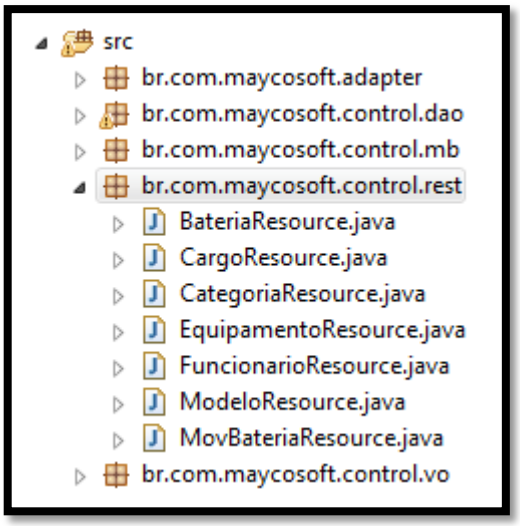

Figura 20 – Localização dos *Web Services*

Os *Web Services* são classes Java normais, porém contendo algumas anotações para expor os serviços necessários. A seguir é apresentado um trecho de código contendo a classe "*CargoResource*" com suas anotações para um melhor entendimento.

```java
package br.com.maycosoft.control.rest;

import java.util.List;

import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.PUT;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;
import javax.xml.bind.JAXBElement;

import br.com.maycosoft.control.dao.CargoDAO;
import br.com.maycosoft.control.vo.CargoVO;

@Path("/cargos")
public class CargoResource {

	@GET
	@Produces(MediaType.APPLICATION_JSON + ";charset=utf-8")
	public List<CargoVO> selectAll() {

		CargoDAO dao = new CargoDAO();
		return dao.selectAll();

	}
```

No trecho de código apresentado podemos observar algumas anotações na prática tais como “*@Path*(“/cargos”)” que aponta o caminho do recurso e a URI que será responsável por receber requisições que neste caso será o “/cargos”, a anotação “*@GET*”, define o recurso como sendo de leitura e por último o “*@Produces*”, especifica o tipo de MIME que o recurso pode produzir e enviar de volta para o cliente.

O *Multipurpose Internet Mail Extensions* (MIME) permite a troca de diferentes arquivos na *Internet*, ou seja, quando ocorre uma transação entre um servidor *Web* e um navegador de *Internet*, o servidor envia primeiro o tipo de MIME do arquivo ao navegador, para ele saber como mostrar o arquivo transmitido (TODD; SZOLKOWSKI, 2003).

O método apresentado no fragmento de código oferece um serviço para listar todos os cargos armazenados no banco de dados, para então serem consumidos pela aplicação *Mobile*.

No trecho a seguir veremos uma classe anotada para receber os dados da aplicação móvel e inseri-las no banco. Este recurso utiliza a API *Java Architecture for XML Binding* (JAXB) que tem a capacidade de mapear a classe objeto em XML e vice versa, possibilitando que os dados sejam armazenados e recuperados na memória em formato XML.

```
@PUT
@Path("/insert")
@Consumes(MediaType.APPLICATION_JSON + ";charset=utf-8")
@Produces(MediaType.APPLICATION_JSON + ";charset=utf-8")
// public CargoVO insert(JAXBElement<CargoVO> element){
public CargoVO insert(CargoVO element) {
	// CargoVO cargo = element.getValue();
	CargoDAO dao = new CargoDAO();
	dao.insert(element);

	return element;
}
```

A classe apresentada no trecho de código pode enviar e receber informações através de JSON. Para passar a trabalhar com XML basta tirar o comentário das duas linhas comentadas e comentar a linha marcada em amarelo.

Na inserção a classe deve possuir a anotação "*@Put*" indicando que é para criar um novo recurso. Também deve ser feita a anotação "*@Consumes*" indicando qual o tipo de MIME que o recurso poderá receber do cliente.

O método apresentado a seguir realiza alteração de cargos. Este por sua vez foi desenvolvido de forma diferente ao método apresentado anteriormente, pois este método esta usando a JAXB, consequentemente trabalhando com arquivo XML e não JSON como mostrado no método de inserção.

```
@POST
@Path("/update")
@Consumes(MediaType.APPLICATION_JSON + ";charset=utf-8")
@Produces(MediaType.APPLICATION_JSON + ";charset=utf-8")
public CargoVO update(JAXBElement<CargoVO> element) {
	// public CargoVO update(CargoVO element) {
	CargoVO cargo = element.getValue();
	CargoDAO dao = new CargoDAO();
	dao.insert(cargo);

	return cargo;

}
```

Neste método foi apresentada uma nova anotação que o "*@Post*". Ele tem a função de atualizar ou criar um novo recurso. As demais anotações são as mesmas do método de inserção, exceto a anotação "*@Path*" modificado para "*/update*" anotando que é um recurso de modificação.

Agora será mostrado a seguir como foi desenvolvido o método responsável por excluir cargos. Na exclusão a anotação "*@Path*" recebe também o código do cargo que será excluído.

```
@DELETE
@Path("/delete/{codigo}")
public CargoVO delete(@PathParam("codigo") int codigo) {
	CargoDAO dao = new CargoDAO();
	dao.delete(codigo);
	CargoVO cargo = new CargoVO();
	cargo.setCodigo(codigo);
	return cargo;

}
```

Podemos observar na exclusão que foi usado o "*@PathParam*" ainda não usado anteriormente. Esta anotação permite o mapeamento de fragmentos de uma URI para o método, ou seja, no exemplo mostrado no método de exclusão de cargos o parâmetro "*/delete/codigo*" da URI, foi passado o código para o argumento do método, para que ele possa manipular e efetuar a exclusão do cargo passado.

Foi demonstrado nesta seção como foi o desenvolvido os Web Services do projeto. Podemos observar no decorrer do texto que foram utilizados todo o conteúdo teórico pesquisado. Com este conteúdo foi possível colocar em prática todo material estudado e consequentemente um melhor entendimento das tecnologias *Java*, utilizadas no processo de construção dos *Web Services*.

6.3. DESENVOLVIMENTO DA APLICAÇÃO MÓVEL

A aplicação móvel foi desenvolvida com o objetivo de coletar as informações obtidas no campo por consequência da movimentação de baterias quando estas são trocadas dos equipamentos, devido há problemas apresentadas na bateria. Como dito anteriormente a aplicação móvel é provida de um banco de dados (*SQLite*) para armazenar as informações.

A aplicação *Mobile* esta estrutura em pacotes para uma maior organização e segue os mesmos conceitos da aplicação *Web*. Alguns pacotes possuem a mesma função dos pacotes da aplicação *Web*, tais como o "*vo"*, o "*dao"*, e o "*adapter"*, portanto não serão necessários maiores explicações, visto que já foram explicados anteriormente. A Figura 21 apresenta os oitos pacotes que foram criados para o desenvolvimento da aplicação *Mobile*.

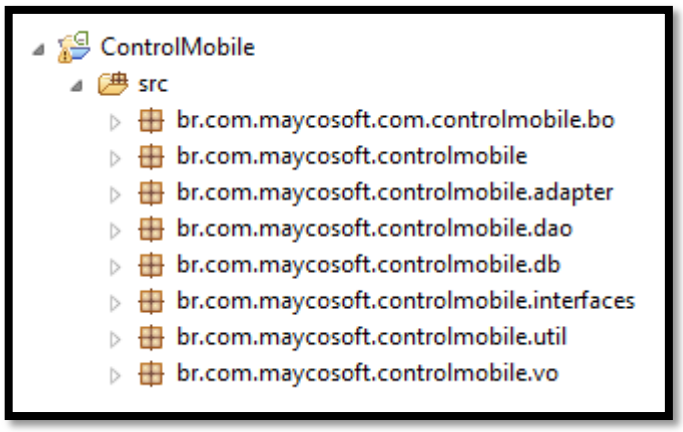

Figura 21 – Pacotes da Aplicação *Mobile*

Como podemos observar na Figura 21, a aplicação móvel possui vários pacotes que seguem alguns padrões de projeto, visando à reutilização de códigos. A seguir será descrito as funções das classes dos principais pacotes mais detalhadamente, mostrando as funcionalidades de cada um.

6.3.1. Pacote Interfaces

As classes localizadas neste pacote são de extrema importância, visto que são responsáveis pela criação das tabelas no banco de dados *SQLite*. No trecho de código a seguir é apresentada a classe responsável por criar a tabela de movimentação de baterias no banco de dados do dispositivo móvel.

```
package br.com.maycosoft.controlmobile.interfaces;

public interface IMovimentacao {

	public static final String TABLE_NAME = "MOVBATERIA";

	public static final String BOLETIM = "BOLETIM";
	public static final String DATA_TROCA = "DATA_TROCA";
	public static final String DESTINO_BATERIA = "DESTINO_BATERIA";
	public static final String ID_EQUIPAMENTO = "ID_EQUIPAMENTO";
	public static final String LADO = "LADO";
	public static final String ID_FUNCIONARIO = "ID_FUNCIONARIO";
	public static final String ID_BATERIA = "ID_BATERIA";
	public static final String OBSERVACAO = "OBSERVACAO";
	public static final String SERVICO = "SERVICO";

	public static final String CREATE = "CREATE TABLE ["+ TABLE_NAME +"] (" +
			" [" + BOLETIM + "] BIGINT  UNIQUE NOT NULL, " +
			" [" + DATA_TROCA + "] DATE  NULL, " +
			" [" + DESTINO_BATERIA +"] VARCHAR(15)  NULL, " +
			" [" + ID_EQUIPAMENTO + "] BIGINT UNIQUE NOT NULL, " +
			" [" + LADO + "] VARCHAR(1)  NULL, " +
			" [" + ID_FUNCIONARIO + "] BIGINT  NOT NULL, " +
			" [" + ID_BATERIA + "] BIGINT  NOT NULL, " +
			" [" + OBSERVACAO + "] VARCHAR(100)  NULL, " +
			" [" + SERVICO + "] VARCHAR(1)  NULL, " +
			" PRIMARY KEY (" + BOLETIM + ") ) ";

   public static final String DROP = "DROP TABLE IF EXISTS [" + TABLE_NAME + "]";
}
```

A classe apresentada no código anterior esta dividida em quatro blocos para facilitar a explicação. O primeiro bloco que é a primeira linha de código esta declarando a variável como o nome da tabela e em seguida esta sendo declaradas as variáveis como o nome das colunas, para depois iniciar a criação das colunas no banco e por último esta excluindo a tabela se caso existir uma tabela com o mesmo nome que esta sendo criada. Devemos criar uma classe para cada tabela que se deseja criar no banco de dados do *Android*.

6.3.2. Business *Object* (BO)

As classes contidas neste pacote têm como função básica encapsular a lógica de negócios para os objetos. É responsável por inserir, alterar, remover dados, entre outras funções.

Para entender melhor as classes contidas neste pacote, antes temos que ter conhecimento das classes contidas no pacote "*br.com.maycosoft.controlMobile.db*". Ele contém duas classes, a "*DataBaseConstants*" e a "*DBHelper*". A classe "*DataBaseConstants*" é bem simples e define o nome e a versão do banco como mostrado a seguir.

```
package br.com.tntedu.escolatnt_v3.db;

public interface DatabaseConstants {

	public static final String DB_NAME = "TCC";

	public static final int VERSION = 2;
}
```

A classe "*DBHelper*" é responsável de um modo geral por atualizar e verificar se já existe um banco criado e em caso negativo invoca um método para criar o banco. A seguir é apresentado o método responsável por criar o banco.

```
@Override
public void onCreate(SQLiteDatabase db) {

            db.execSQL("PRAGMA foreign_keys = ON");
           db.execSQL(IMovimentacao.CREATE);

      for (int versaoAtual = 2; versaoAtual <= DatabaseConstants.VERSION;
            versaoAtual++) {
            atualizarBaseDados(db, versaoAtual);
      }
}
```

Explicado as classes do pacote "*br.com.maycosoft.controlMobile.db*" fica mais fácil o entendimento do trecho de código a seguir, visto que nele cria-se uma instancia da classe "*DBHelper*" como mostrado na linha grifada em amarelo.

```
public boolean inserir(Context context, MovimentacaoVO movimentacao) throws
Exception {
            DBHelper helper = new DBHelper(context);
            SQLiteDatabase db = helper.open();

            try {
                  db.beginTransaction();
                  dao.inserir(db, movimentacao);
                  db.setTransactionSuccessful();

            } catch (Exception e) {
                  throw e;
            } finally {
                  db.endTransaction();
                  db.close();
            }

            return true;

      }
```

No trecho de código apresentado podemos observar que temos o método responsável por inserir informações no banco. Este método faz todo o processo de persistência de dados no banco.

6.3.3. Pacote *Adapter*

No *Android* quando precisamos preencher componente como *ListView*, *Spinner* e *GridView*, utiliza-se o conceito de *Adapters*. Esta classe é responsável por adaptar uma lista de objetos para uma lista com elementos de interface gráfica, como por exemplo uma *ListView*. A seguir é mostrado um trecho de código da classe "*movimentacaoAdapter*" do projeto *Mobile*.

```
public class MovimentacaoAdapter extends BaseAdapter {

	private LayoutInflater inflater = null;
	private List<MovimentacaoVO> movimentacoes = null;
	private ViewHolder holder = null;

	static class ViewHolder {
		public TextView lblEquipamento;
		public TextView lblModeloEquipto;
		public TextView lblNumFogoBat;
	}

	public MovimentacaoAdapter(Context context, List<MovimentacaoVO>
movimentacoes) {
		inflater = LayoutInflater.from(context);
		this.movimentacoes = movimentacoes;
	}

	public int getCount() {
		return movimentacoes.size();
	}

	public Object getItem(int position) {
		return movimentacoes.get(position);
	}

	public long getItemId(int position) {
		return movimentacoes.get(position).getBoletim();
	}

	public View getView(int position, View convertView, ViewGroup parent) {

		if (convertView == null) {
			convertView =
inflater.inflate(R.layout.tela_movimentacao_adapter, null);

			holder = new ViewHolder();
			holder.lblEquipamento = (TextView)
```

```
convertView.findViewById(R.id.lblEquipamento);
                    holder.lblModeloEquipto = (TextView)
convertView.findViewById(R.id.lblModeloEquipto);
                    holder.lblNumFogoBat = (TextView)
convertView.findViewById(R.id.lblNumFogoBat);

                    convertView.setTag(holder);

       } else {
                    holder = (ViewHolder) convertView.getTag();
                }
              MovimentacaoVO movimentacao = movimentacoes.get(position);

      holder.lblEquipamento.setText(Long.toString(movimentacao.getEquipamento()));
      holder.lblModeloEquipto.setText(movimentacao.getModeloEquipto());
      holder.lblNumFogoBat.setText(movimentacao.getNumFogoBat());

              return convertView;
      }
}
```

É possível observar no trecho de código apresentado, que a classe "movimentacao*Adapter*" estende da *BaseAdapter*. Quando é criado uma classe *adapter* é necessário implementar quatro métodos:

- ***getCount*:** responsável por retornar a quantidade de linhas que o adaptador representa, ou seja, retorna o tamanho de uma determinada lista;
- ***getItem*:** tem a responsabilidade de acessar o objeto que o adaptador esta representando e retornar o objeto da posição da lista;
- ***getItemId*:** este método é responsável por retornar o identificador do objeto do objeto passado por parâmetro;
- ***getView*:** é o método mais importante, pois é responsável por pegar o objeto da lista, carregar o arquivo de *layout* e atribuir os valores dos atributos do objeto para a *view*, que representará a linha da lista.

Na linha de código grifada em amarelo é possível perceber que para carregar o arquivo de *layout* utilizamos a classe *LayoutInflater*. Depois de carregado o *layout*, é retornado um objeto *View* que representa a árvore *Views* definida no arquivo XML, ou seja, é passado os valores dos atributos para os *TextViews*.

6.3.4. Pacotes de *Activity*

As *activities* são os componentes mais utilizados em uma aplicação *Android*, pois fazem o gerenciamento dos eventos da interface gráfica, respondendo aos eventos solicitados pelo usuário e pela aplicação. Na Figura 22 são apresentadas as *activities* da aplicação móvel.

Figura 22 – Activity do Projeto *Mobile*

Na Figura 22 foram apresentadas as principais *activities* da aplicação. A seguir será apresentado um fragmento de código responsável por uma interface gráfica.

```
public class MovimentacaoActivity extends Activity {

	private RadioGroup rdgServico = null;
	private EditText txtDataTroca = null;
	private EditText txtMatricula = null;
	private EditText txtNome = null;
	private EditText txtEquipamento = null;
	private EditText txtModelo = null;
	private EditText txtLado = null;
	private EditText txtNumFogoBat = null;
	private EditText txtNumSerieBat = null;
	private EditText txtMarcaBateria = null;
	private EditText txtAmperagem = null;
	private EditText txtDestino = null;
	private EditText txtObservacao = null;
	private MovimentacaoVO movimentacao = null;
	private Button btnSalvar = null;
	private boolean isUpdate = false;
```

```
	private MovimentacaoBO movimentacaoBO = new MovimentacaoBO();
	private static NumberFormat format = new DecimalFormat("00");

	@Override
	protected void onCreate(Bundle savedInstanceState) {

		super.onCreate(savedInstanceState);
		setContentView(R.layout.tela_movimentacao);

		rdgServico = (RadioGroup) findViewById(R.id.rdgServico);
		txtDataTroca = (EditText) findViewById(R.id.txtDataTroca);
		txtMatricula = (EditText) findViewById(R.id.txtMatricula);
		txtNome = (EditText) findViewById(R.id.txtNome);
		txtEquipamento = (EditText) findViewById(R.id.txtEquipamento);
		txtModelo = (EditText) findViewById(R.id.txtModelo);
		txtLado = (EditText) findViewById(R.id.txtLado);
		txtNumFogoBat = (EditText) findViewById(R.id.txtNumFogo);
		txtNumSerieBat = (EditText) findViewById(R.id.txtNumSerie);
		txtMarcaBateria = (EditText) findViewById(R.id.txtMarcaBateria);
		txtAmperagem = (EditText) findViewById(R.id.txtAmperagem);
		txtDestino = (EditText) findViewById(R.id.txtDestino);
		txtObservacao = (EditText) findViewById(R.id.txtObservacao);

		btnSalvar = (Button) findViewById(R.id.btnSalvar);

		txtDataTroca.setOnClickListener(new View.OnClickListener() {

			public void onClick(View arg0) {
				selecionarData();
			}
		});

		btnSalvar.setOnClickListener(new View.OnClickListener() {

			public void onClick(View view) {
				salvar();
			}
		});
		Bundle parametros = getIntent().getExtras();

		if (parametros != null && parametros.containsKey("movimentacao")) {

			movimentacao = (MovimentacaoVO)
parametros.getSerializable("movimentacao");
			isUpdate = true;
			carregarMovimentacao();

		}else{
			movimentacao = new MovimentacaoVO();
		}

	}

	 // Foram omitidos o restante do código
```

O trecho de código apresentado gerencia a tela referente à movimentação de baterias, nela é possível inserir todos os dados necessários para a troca de baterias e salvá-los no banco de dados da aplicação móvel.

É importante frisar que a activities contém um ciclo de vida e neste ciclo existem alguns métodos predefinidos, tais como o *onCreate()* que foi apresentado no trecho de código, o *onResume()*, *onPause()* e o *onStop()*.

6.3.5. Interface Gráfica da Aplicação

Despois de mostrado toda a parte do processo de desenvolvimento da aplicação *Mobile*, agora será mostrado as interfaces gráficas. A aplicação móvel é constituida basicamente de três telas que serão apresentada a seguir.

A Figura 23 apresenta a tela de login, onde o usuário é obrigado a informar a matrícula e a senha dele para acessar as funcionalidades da aplicação.

Figura 23 – Tela de Autenticação de Usuário

Outra tela muito importante é a tela de sincronização dos dados, onde o usuário tem a funcionalidade de sincronizar os dados com o *Web Services*. Nesta tela ele tem a opção de escolher os dados que deseja enviar ou atualizar. A Figura 24 apresenta a interface gráfica da tela de sincronização com as opções de escolha.

Figura 24 – Tela de Sincronização

A tela de movimentação de baterias é a mais extensa devido ao grande número de informações exigidas na troca de uma bateria. Nela o usuário é obrigado a informar todos os dados necessários para que seja efetuada a troca da bateria num

determinado equipamento. Estas informações ficam salvas no banco de dados do dispositivo móvel, para posteriormente serem enviadas para a aplicação *Web*, que após assegurar que o envio foi concluído com sucesso as movimentações são apagadas do dispositivo móvel. A Figura 25 apresenta a interface responsável pela movimentação de baterias, que devido à extensão da tela, foi usado o componente "*scrollView"*, da interface gráfica do *Android*.

Figura 25 – Tela de Movimentação de Baterias

Hoje devido ao grande número de dispositivos móveis utilizando a plataforma *Google Android* e os usuários cada vez mais conhecedor e familiarizado com as funcionalidades que esta plataforma oferece, torna mais fácil ao usuário interagir com novas aplicações do estilo que foi apresentada, ou seja, a interface é bem simples e intuitiva, fazendo que o usuário não tenha grandes dificuldades para utilizar a aplicação, bastando apenas de um treinamento para mostra-lo como será o procedimento que ele deve seguir no campo para efetuar as trocas de baterias.

6.4. PADRÕES DE PROJETO

No desenvolvimento deste projeto foram utilizados alguns padrões de projetos visando à organização, reutilização e a preparação para o mercado de trabalho.

Os padrões de projeto descrevem soluções de um modo geral para reutilização de códigos recorrente no desenvolvimento de aplicações orientadas a objetos. Nesta descrição é definido um modelo de como resolver o problema a ser tratado e que pode ser usado em muitas outras situações diferentes. Normalmente os padrões de projeto apresentam definições e interações entre classes ou objetos, sem determinar detalhes da classe ou objetos envolvidos (GAMMA et al., 2006).

No projeto foram abordados vários padrões de projeto, tais como:

- ***Value Object (VO)*:** é um padrão para transferência agregada de dados entre classes diferentes (SAMPAIO, 2007);
- ***Data Acess Object*:** é um padrão responsável por persistir os dados no banco, separando as regras de negócio das regras de acesso ao banco de dados;
- ***Factory Method*:** consiste em apresentar uma proposta para atribuir responsabilidade para criação de objetos, ou seja, permite que uma classe confie a instanciação para as subclasses (SHALLOWAY; TROTT, 2002).
- ***Adapter*:** como o próprio nome diz, ele provê mecanismos para adaptar duas interfaces incompatíveis. Na plataforma *Android* este padrão serve para adaptar uma lista de objetos para uma lista de elementos de interface gráfica, como, por exemplo, as linhas de uma *ListView* (HORSTMANN, 2006);
- ***Business Object (BO)*:** sua função básica é encapsular as regras de negócio para um objeto, ou seja, separar os dados e a lógica de negócio de um determinado objeto.

Acima foram descritos alguns padrões de projeto que no decorrer do texto foram citados e agora explicados melhor os conceitos de cada um.

Neste capítulo foi apresentado todo o processo de desenvolvimento da aplicação *Mobile*, colocando em prática o conteúdo teórico pesquisado, mostrando toda à estrutura disposta em pacotes para manter a organização dos códigos, além de ter sido apresenta boa parte dos códigos que foram necessários no desenvolvimento da aplicação. Foi apresentado também o desenvolvimento do banco de dados para a aplicação móvel, visto a necessidade em se manter os dados armazenados devido a não cobertura total do sinal de *Internet* no campo.

7 – CONCLUSÃO

Pesquisa recente divulgada pela *Strategy Analytics* mostrou que o sistema operacional *Android* continua crescendo. No segundo trimestre de 2013, o *Android* detinha 75% dos consumidores internacionais, já no terceiro trimestre deste ano, esse número aumentou para 81,3% e em segundo lugar aparece o iOS da *Apple* com 13,4%.

Devido a essa grande aceitação por parte dos usuários pelo sistema operacional *Android*, surge grandes oportunidades de melhorias para as empresas que hoje utilizam informações geradas fora do ambiente interno da empresa, mas que precisam ser agregadas as informações internas para assim ter um controle mais efetivo. Hoje boa parte das empresas, usam formulários para coletar estes dados, ocasionando muita perda de informações e consequentemente prejudicando a empresa na elaboração de melhorias futuras.

Este trabalho teve como objetivo desenvolver uma pesquisa teórica, com o intuito de adquirir conhecimentos sobre a arquitetura orientada a recursos, bem como o desenvolvimento de *Web Services RESTful*. Juntamente a estes estudos, foi desenvolvido um estudo de caso, onde foi possível aplicar os conceitos adquiridos na pesquisa teórica, e possibilitando o desenvolvimento de um ambiente orientado a recursos, capaz de interagir com uma aplicação desenvolvida na plataforma *Google Android*, onde esta passa a consumir os serviços expostos.

Portanto o objetivo deste trabalho foi alcançado, pois foi desenvolvimento uma aplicação *Web*, capaz de gerenciar toda a parte lógica do negócio e juntamente a esse foi desenvolvido um ambiente orientado a recursos, responsável pela exposição dos serviços que serão acessadas e consumidas pela aplicação *Mobile*.

Depois da pesquisa teórica juntamente com as aplicações desenvolvidas neste trabalho, fora possível perceber que outras aplicações podem ser desenvolvidas tendo como base a mesma arquitetura utilizada neste trabalho.

Com o trabalho realizado conclui-se também, que esta plataforma pode fazer muito mais do que simplesmente acessar redes sociais, contas bancárias, páginas de notícias entre outras infinidades de opções disponíveis. Esta plataforma pode contribuir muito nos processos externos das empresas, possibilitando a coleta de informações fora da empresa para posteriormente serem enviadas para os sistemas das empresas através da *Internet*, visto que praticamente todos os usuários de *smartphones* possuem pacotes de acesso a *Internet*.

7.1. TRABALHOS FUTUROS

Para trabalhos futuros, serão desenvolvidos relatórios para auxiliar no controle de baterias, visando um melhor aproveitamento da garantia assim como um processo de sincronização automática do aplicativo *Web* com o *Mobile* quando ocorrer um novo cadastro, seja de equipamento, funcionário, cargo ou bateria. Outro recurso a ser desenvolvido, será o armazenamento das coordenadas geométricas do local onde esta sendo feito a troca da bateria, através do GPS do dispositivo móvel.

Continuando no segmento de usina de cana de açúcar, outros módulos podem ser desenvolvidos para a coleta de dados no campo, tais como apontamento mecânico, apontamento de movimentação de pneus e lubrificação de equipamentos que, atualmente, utilizam formulários preenchidos manualmente para posteriormente serem digitados.

Ainda se pode desenvolver um módulo para gerenciar a coleta de informações geradas pelas pequenas intervenções realizadas pelo próprio usuário no equipamento. Atualmente, estas informações são passadas por rádio amador a um funcionário da empresa que, posteriormente, faz digitação desses dados.

Como se pode perceber, a arquitetura utilizada neste trabalho abre caminho para várias outras possibilidades de trabalhos futuros, não só de usina de cana de açúcar, mas também para qualquer outra empresa que depende de informações geradas de forma externa ao ambiente de trabalho.

REFERÊNCIAS

ABINADER, Jorge Abílio; LINS, Rafael Dueire. **Web Services em Java**. 1. ed. Rio de Janeiro: Brasport Livros e Multimídia Ltda., 2006.

ABLESON, W. Frank; COLLINS, Charlie; SEN, Robi. **Unlocking Android** – A Developer's Guide. 1. ed. Greenwich: Manning, 2009.

ALONSO, Gustavo; CASATI, Fábio; KUNO, Harumi; MACHIRAJU, Vijay. **Web Services** – Concepts, Architectures and Applications. 1. ed. Germany: Springer, 2004.

AVELAREDUARTE. **Serviços Web (Web Service).** Disponível em: <http://www.avellareduarte.com.br/projeto/conceitos/webservices/webservices.htm>. Acesso em: 06 jul. 2013.

BARROS, Thiago. **Android domina mercado brasileiro; Windows Phone mais Expressivo que iPhone.** Techtudo Notícias. Disponível em: <http://www.techtudo.com.br/noticias/noticia/2012/09/android-domina-mercado-brasileiro-windows-phone-mais-expressivo-que-iphone.html>. Acesso em 10 mar. 2013.

BURKE, Bill. **RESTful Java With JAX-RS.** 1 ed. Sebastopol: O' Reilly Media, Inc, 2010.

BROEMMER, Darren. **J2EE Best Practices** – Java Design Patterns, Automation, and Performance. 1. ed. Indianápolis: Wiley Publishing, 2003

CAELUM, Ensino e Inovação. **Java para Desenvolvimento Web.** 1. ed. São Paulo: Caelum Ensino e Inovação, 2012.

CERAMI, Etban. **Web Services Essentials** – Distributed Applications With XM – RPC, SOAP, UDDI e WSDL. 1. ed. Sebastopol: O' Reilly Media, 2002.

FARIA, Alessandro de Oliveira. Programe seu Andróide. **Linux Magazine**, v.1, n.43, 2008, p. 73-77.

FARTO, Guilherme de Cleva. **Abordagem Orientada a Serviços para Implementação de um Aplicativo Google Android.** 2010, 83p. Monografia de Conclusão de Curso – Instituto Municipal de Ensino Superior de Assis, São Paulo, Assis, 2010.

FIELDING, Roy Thomas. **Architectural Styles and the Design of Network-based Software Architectures.** 2000. 180p. Tese (doutorado) – University of California, California, Irvine, 2000.

FRANÇA, Thiago C.; PIRES, Paulo F.; PIRMEZ, Luci; DELICATO, Flávia C.; FARIAS, Cláudio. Web das coisas: conectando dispositivos físicos ao mundo digital. In: SIMPÓSIO BRASILEIRO REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS, 29, 2011, Campo Grande, Brasil. **Anais do XXIX Simpósio Brasileiro de Redes de Computadores e Sistemas Distribuídos**, v. 1, junho, 2011. p. 103 – 150.

GAMMA, Erich; HELM, Richard; JOHNSON, Ralph; VLISSIDES, John. **Padrões de Projeto** – Soluções Reutilizáveis de Software Orientado a Objetos. 1. ed. Tradução de Luiz A. Meirelles Salgado. Porto Alegre: Bookman Companhia Editora, 2006.

GARGENTA, Marko. **Learning Android** – Bulding Applications for the Android Market. 1. ed. Sebastopol: O' Reilly Media, 2011.

GOMES, Fábio de Jesus Lima; SOUZA, Manoel Taenan Ferreira; ARAÚJO, Rafael Madureira Lins. Desenvolvimento de aplicações para a plataforma Google Android. In: ENCONTRO REGIONAL DE INFORMÁTICA, 5, 2011, Terezina, Brasil. **Anais do V Encontro Regional de Informática**, v.1, novembro, 2011. p.100 – 123.

GONÇALVES, Antonio. **Beginning Java EE™ 6 Platform With GlassFish™ 3** – From Novice to Professional. 2. ed. New York: Apress, 2010.

HORSTMANN, Cay. **Padrões e Projetos Orientados a Objetos.** 2. ed. Porto Alegre: Bookman Companhia Editora, 2006.

JENDROCK, Eric; EVANS, Ian; GOLLAPUDI, Devika; HAASE, Kim; SRIVATHSA, Chinmayee. **The Java EE 6 Tutorial: Basic Concepts** – Java Series Enterprise Edition. 4 ed. Boston: Pearson Education, 2010.

JENDROCK, Eric; NAVARRO, Ricardo Cervera; EVANS, Ian; GOLLAPUDI, Devika; HAASE, Kim; MARKITO, William; SRIVATHSA, Chinmayee. **The Java EE 6 Tutorial.** 1 ed. Rewood City: Oracle USA, 2013.

LECHETA, Ricardo R. **Google Android** – Aprenda a criar aplicações para dispositivos móveis com o Android SDK. 2. ed. São Paulo: Novatec Editora, 2010.

LECHETA, Ricardo R. **Google Android para Tablets** – Aprenda a desenvolver aplicações para o Android – De smartphones a tablets. 1. ed. São Paulo: Novatec Editora, 2012.

LIMA, Jean Carlos Rosário. **Web Services (SOAP X REST).** 2012. 41p. Trabalho de Conclusão de Curso. Faculdade de Tecnologia de São Paulo, São Paulo, 2012.

MORO, Tharcis Dal; DORNELES, Carina F.; REBONATTO, Marcelo Trindade. Web services WS-* versus Web Services REST. **Reic – Revista de Iniciação Cientifica**, v. 11, n. 1, março, 2011, p.39-51.

OLIVEIRA, Eric C. M. **Conhecendo a plataforma J2EE – Um Breve Overview.** Desenvolvimento – Java – Linha de código. Disponível em: <http://www.linhadecodigo.com.br/artigo/333/conhecendo-a-plataforma-j2ee-um-breve-overview.aspx>. Acesso em: 10 jun. 2013.

OLIVEIRA, Ricardo Ramos. **Avaliação de Manutenibilidade Entre as Abordagens de Web Services RESTful e SOAP-WSDL.** 2010. 99p. Dissertação (mestrado) – Instituto de Ciências Matemáticas e de Computação – ICMC-USP, São Paulo, São Carlos, 2012.

OPEN HANDSET ALLIANCE. Disponível em: <http://www.openhandsetalliance.com>. Acesso em: 20 jul. 2013.

PENCHIKALA, Srini. **Java EE 6 – Suporte a REST com Anotações no JAX-RS 1.1.** INFOQ – Traduzido por Renato Marco. Disponível em: <http://www.infoq.com/br/news/2010/02/javaee6-rest>. Acesso em: 09 jul. 2013.

PEREIRA, Lúcio Camilo Oliva; SILVA, Michel Lourenço da Silva. **Android para Desenvolvedores.** 1. ed. Rio de Janeiro: Brasport Livros e Multimídia Ltda., 2009.

POLÔNIA, Pablo Valério. **Proposta de Arquitetura Orientada a Recursos para Scada na Web.** 2011. 145p. Dissertação (mestrado) – Universidade Federal de Santa Catarina – Centro Tecnológico, Santa Catarina, Florianópolis, 2011.

PULIER, Eric; TAYLOR, Hugh. **Compreendendo SOA Corporativa**. 1. ed. Tradução de Marcelo Trannin Machado. Rio de Janeiro: Ciência Moderna, 2008.

RENOUF, Colin. **Pro (IBM) WebSphere Application Server 7 Internals.** This book is for Java EE developers, architects, and administrators who want to Know how WebSphere Application Server 7 works under the covers. 1. ed. New York: Apress, 2009.

RICHARDSON, Leonard; RUBY, Sam. **RESTful Web Services** – Web Services For The Real World. 1. ed. United States of America: O' Reilly Media, 2007.

ROVARIS, Felipe Machado. **Automação na Cotação de Livros Utilizando Web Service.** Universidade Federal de Santa Catarina – UFSC – Departamento de Informática e Estatística. Disponível em <https://projetos.inf.ufsc.br/arquivos_projetos/projeto_583/artigo%20lipous.pdf>. Acesso em: 25 fev. 2013.

SAMPAIO, Cleuton. **Guia do Java Enterprise Edition 5** – Desenvolvendo Aplicações Corporativas. 1 ed. Rio de Janeiro: Brasport Livros e Multimídia Ltda., 2007.

SAMPAIO, Cleuton. **Java Enterprise Edition 6** – Desenvolvendo Aplicações Corporativas. 1 ed. Rio de Janeiro: Brasport Livros e Multimídia Ltda., 2011.

SILVESTRE, Erich; POLÔNIA, Pablo Valério. **Uma Aplicação da Arquitetura Orientada a Recursos.** 2008. 250p. Trabalho de Conclusão de Curso. Engenharia de Controle e Automação – UFSC – Universidade Federal de Santa Catarina, Santa Catarina, 2008.

SOAINSTITUDE.ORG. **A Peer to Peer Exchange for Service Oriented Architecture Professionals.** Disponível em: <http://www.soainstitute.org/resources/articles/resource-oriented-architecture>. Acesso em: 27 jun. 2013.

SOUZA, Marcelo Soares. **Rejunte Middleware para Integração e Compartilhamento de Acervos Culturais na Web 2.0.** 2010, 49p. Monografia de Conclusão de Curso – Universidade Católica de Salvador – Instituto de Ciências Exatas e Tecnológicas, Bahia, Salvador, 2010.

SHALLOWAY, Alan; TROTT, James R.. **Explicando Padrões de Projeto** – Uma Nova Perspectiva em Projeto Orientado a Objeto. 1 ed. Porto Alegre: Bookman Companhia Editora, 2002.

TELECO, Inteligência em Telecomunicações. **Estatísticas de Celular no Mundo.** Disponível em: <http://www.teleco.com.br/pais/celular.asp>. Acesso em: 17 jul. 2013.

TODD, Nick; SZOLKOWSKI, Mark. **JavaServer Pages O Guia do Desenvolvedor** – O Guia Definitivo para Desenvolver Aplicações com JavaServer Pages 2.0. 1. ed. Rio de Janeiro: Elsevier Brasil, 2003.

TOSIN, Carlos Eduardo Gusso. **A Plataforma Android.** Softblue Cursos Online. Disponível em: <http://www.softblue.com.br/blog/home/postid/11/CONHECENDO+O+ANDROID>. Acesso em: 10 ago. 2013.

VOHRA, Deepak. **EJB 3.0 Database Persistence With Oracle Fusion Middleware 11g.** 1 ed. Birmingham: Packt Publishing Ltda, 2010.