CENTRO UNIVERSITÁRIO LUTERANO DE PALMAS
ULBRA
COMUNIDADE EVANGÉLICA LUTERANA "SÃO PAULO"
Recredenciado pela Portaria Ministerial nº 3.607 - D.O.U. nº 202 de 20/10/2005

FABRÍCIO CORREIA DA SILVA

DESENVOLVIMENTO DE APLICATIVO EM ANDROID PARA AUXÍLIO DA POLÍCIA MILITAR EM BLITZ

Palmas

2012

FABRÍCIO CORREIA DA SILVA

DESENVOLVIMENTO DE APLICATIVO EM ANDROID PARA AUXÍLIO DA POLÍCIA MILITAR EM BLITZ

Trabalho apresentado como requisito parcial da disciplina Trabalho de Conclusão de Curso (TCC) do curso de Sistemas de Informação, orientado pelo Professor Mestre Edeílson Milhomem da Silva.

Palmas

2012

FABRÍCIO CORREIA DA SILVA

DESENVOLVIMENTO DE APLICATIVO EM ANDROID PARA AUXÍLIO DA POLÍCIA MILITAR EM BLITZ

Trabalho apresentado como requisito parcial da disciplina Trabalho de Conclusão de Curso (TCC) do curso de Sistemas de Informação, orientado pelo Professor Mestre Edeílson Milhomem da Silva.

Aprovada em xxxxxxx de 2012.

BANCA EXAMINADORA

_____

Prof. M.Sc. Edeílson Milhomem da Silva
Centro Universitário Luterano de Palmas

_____

Prof. M.Sc. Jackson Gomes de Souza
Centro Universitário Luterano de Palmas

_____

Profª. M.Sc. Madianita Bogo
Centro Universitário Luterano de Palmas

Palmas
2012

AGRADECIMENTOS

Agradeço a todos os que duvidaram que um dia eu fosse me formar. Isto me trouxe força e ímpeto para a conclusão deste trabalho. Agradeço a Deus por me dar obstáculos, pois assim pude crescer como pessoa e profissional. Agradeço a todos que me ajudaram neste trabalho, principalmente à Elizabeth “manola” Silva, que não me deixou desanimar. Ela desempenhou o papel de minha agenda, lembrando-me todos os dias que eu teria que fazer o TCC para me formar. Agradeço a esta pessoa maravilhosa, que em muitos momentos achei irritante, mas que fez isto para o meu bem. Agradeço aos professores, sempre muito gentis e acolhedores, deixando-me bastante a vontade. Enfim, agradeço a todo o caos existente, pois não estaria aqui se não houvesse todos os problemas que um dia tentaram me fazer desistir.

SUMÁRIO

RESUMO

A análise de dados de veículos e condutores no Estado do Tocantins tem a responsabilidade de verificar se o contribuinte está de acordo com as leis que regem o Estado, sendo de responsabilidade da Polícia Militar a aferição destes dados. Atualmente, os dados analisados são oriundos de documentos impressos de porte obrigatória do condutor do veículo, não contendo todas as informações necessárias acerca de veículos e condutores, havendo também a possibilidade de falsificação dos mesmos.

O presente trabalho tem como objetivo o desenvolvimento de um *software* para dispositivos móveis que atenda às necessidades supracitadas, acessando dados em tempo real no Detran-TO. Isso possibilita a verificação de informações atualizadas, permitindo à Polícia Militar a checagem de ocorrências de roubo/furto, apreensão de veículo e existência de débitos ou penalidades.

PALAVRAS-CHAVE: Detran, Polícia Militar, Dispositivos Móveis.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE LISTAGENS

LISTA DE ABREVIATURAS

AD	*Active Directory*
ADT	*Android Development Tools*
AJAX	*Asynchronous JavaScript and XML*
API	*Application Programming Interface*
ATTM	Agência de Trânsito, Transporte e Mobilidade
BIN	Base de Informações Nacional
CEP	Código de Endereçamento Postal
CIRETRAN	Circunscrição Regional de Trânsito
CNH	Carteira Nacional de Habilitação
CNPJ	Cadastro Nacional de Pessoa Jurídica
CPF	Cadastro de Pessoa Física
CPU	*Central Processing Unit*
CRLV	Certificado de Registro e Licenciamento de Veículos
CSS	*Cascading Style Sheet*
Detran	Departamento de Trânsito
DOM	*Document Object Model*
DPVAT	Danos Pessoais Causados por Veículo Automotores de Via Terrestre
EDGE	*Enhanced Data rates for GSM Evolution*
GPS	*Global Positioning Interface*
GSM	*Groupe Spécial Mobile*
HTTP	*HyperText Transfer Protocol*
IDE	*Integrated Development Environment*
IPVA	Imposto sobre a Propriedade de Veículos Automotores
OHA	*Open Handset Allience*
RAM	R*andom Access Memory*
RPC	*Remote Procedure Call*
SDK	*Software Development Kit*
SMS	*Short Message Service*
SOAP	*Simple Object Access Protocol*
TCP	*Transmission Control Protocol*

UDDI	*Universal Description, Discover and Integration*
URI	*Unique Resource Identifier*
URL	*Uniform Resource Locator*
UUID	*Universally Unique Identifier*
WSDL	*Web Services Description Language*
XML	*Extensible Markup Language*

1 INTRODUÇÃO

A fiscalização e o controle de veículos e condutores é uma das tarefas da Polícia Militar do Estado do Tocantins. Através desta fiscalização, é possível verificar e aferir a documentação de condutores e veículos. A realização destas verificações é importante para credenciar condutores e veículos a trafegarem por vias públicas, atestando que estão conforme as leis perante o Estado do Tocantins e a União. A aferição, atualmente, é feita através da verificação dos documentos CNH, Carteira Nacional de Habilitação, e CRLV, Certificado de Registro e Licenciamento de Veículos. O que dificulta a verificação destes documentos por parte da Polícia Militar é a grande falta de informação destes documentos, além de serem passíveis de falsificação e não possuir informações em tempo real, tais como ocorrência de roubo/furto em aberto, solicitação de apreensão de veículo ou ordens judiciais que impossibilitam o tráfego do veículo e/ou condutor por vias públicas.

A partir deste contexto, pensou-se na elaboração de um aplicativo para auxiliar a Polícia Militar em sua fiscalização em blitz, tendo como base os seguintes fatores:

- O Detran-TO, Departamento Nacional de Trânsito do Estado do Tocantins, em suas inúmeras funções, tem o dever de regularizar e verificar as conformidades dos veículos automotores que trafegam pelo Estado. Estas verificações são realizadas através de blitz, postos policiais e rondas feitas pelas polícias diariamente. Nestes momentos, é verificado o estado do carro com relação as suas conformidades para trafegar, como vida útil do pneu, setas em perfeito estado, condições atuais do veículo, dentre outros; além dos documentos obrigatórios por parte do proprietário e do veículo;
- O veículo deve estar com suas obrigações perante o Detran-TO em dia, como a quitação do licenciamento anual, o IPVA (Imposto sobre a Propriedade de Veículos Automotores) e o DPVAT (Danos Pessoais Causados por Veículos Automotores de Via Terrestre), além de não conter multas registradas não quitadas ou alguma restrição que impossibilite o veículo de trafegar como existência de ocorrência de roubo/furto destinada ao veículo;
- O condutor também deve ter suas obrigações perante o Detran-TO regularizadas, como ter em sua posse a CNH atual, dando-lhe o direito de trafegar pelas vias nacionais;

- O controle feito pelos policiais do Estado do Tocantins é realizado única e exclusivamente pela verificação dos documentos que estão em posse do condutor do veículo, sendo verificada a CNH do condutor e o CRLV do veículo, não contendo todas as informações pertinentes e necessárias para auxiliar a Polícia Militar em suas tarefas;
- Os pontos acumulados por infrações destinadas ao condutor e a existência de ocorrência de roubo/furto não são verificados por não ser possível um fácil acesso às informações em tempo real, resultando na liberação do condutor e do veículo sem a verificação dessas premissas. A verificação de roubo/furto de veículo só é realizada quando a polícia já tem conhecimento prévio e esteja à procura de um determinado veículo, onde todos os policiais são alertados através de comunicação via rádio. Os veículos de outro Estado que, por ventura, ingressam nos limites do Tocantins, trafegam livremente sem possuir um sistema que verifique o Estado atual de tal veículo junto à BIN (Base de Informações Nacional).

Dentro dos relatos, problemas e dificuldades detectados quanto aos serviços de fiscalização prestados pela Polícia Militar no Estado do Tocantins, este trabalho tem como objetivo o desenvolvimento de uma aplicação que acessará os dados do condutor e do veículo em tempo real, obtendo informações não somente na base local do Detran no Estado do Tocantins, mas também possuindo a característica de verificar os dados na Base Nacional, contemplando além dos veículos do Tocantins, os veículos de toda a União.

A aplicação auxiliará os policiais na verificação da legitimidade dos documentos que estão em posse do condutor, bem como as pendências existentes no Detran com relação ao condutor e ao veículo. Serão apresentadas as informações referentes a roubo/furto, mandado de apreensão, características do veículo, características da CNH do condutor, bem como todas as informações reais referentes ao veículo e condutor.

Documentos falsificados poderão ser checados junto à aplicação, apresentando ao policial os dados reais do condutor e do veículo, sendo possível a averiguação de todas as características reais dos documentos e as características físicas do veículo. Essa averiguação será realizada a partir da comparação das informações buscadas pelo aplicativo em tempo real na base de dados do Detran-TO com as informações apresentadas nos documentos de porte do proprietário.

Para que a aplicação fosse de fácil manuseio e acesso, decidiu-se por uma aplicação que pudesse ser executada em plataformas compactas e simples, que pudesse ser utilizada a qualquer momento e em qualquer local, possuindo agilidade para auxiliar a Polícia Militar de forma prática e eficiente. Escolheu-se utilização de *smartphones* e *tablets*, visto que o

mercado de telefones celulares vem crescendo de forma rápida, em especial aparelhos *smartphones*, que são telefones celulares que possuem recursos avançados e particulares, tais como tela sensível ao toque, sistema operacional capaz de conter aplicativos desenvolvidos por grandes empresas ou por pessoas comuns, além do alto poder de processamento, muitas vezes superior aos computadores convencionais. Em busca de atingir esse mercado, diversas empresas, como o Google, uniram-se a fim de formar a OHA (*Open Handset Allience*) e lançar a plataforma Android (OHA, 2012a, tradução nossa).

A plataforma Android é um projeto de código aberto, que suporta tecnologias oferecidas nos *smartphones* mais modernos como tela sensível a toque (*touchscreen*) e GPS (*Global Positioning System*). Segundo OHA (2012b, tradução nossa), esta plataforma é composta por um conjunto de *softwares* destinados aos dispositivos móveis, que abrange desde sistema operacional a aplicativos com uma vasta gama de auxílio ao usuário, como *softwares* de bancos, acesso a Internet, mapas com localização do usuário em tempo real, entre outros.

Uma peculiaridade interessante do Android é permitir que aplicativos sejam desenvolvidos por terceiros, existindo a possibilidade de disponibilizá-lo para *download* na loja virtual Google Play, sendo esta uma loja criada para servir de repositórios de aplicações Android, onde o desenvolvedor da aplicação escolhe um preço a ser cobrado por cada *download* ou disponibiliza gratuitamente o acesso à aplicação. O fato de ser um Sistema Operacional desenvolvido pelo Google permite uma total integração com todos os serviços desta empresa, desde acessos a mapas, controles de voz, tradução simultânea de textos para determinados idiomas, compras com cartões de créditos através de conta Google, dentre outros.

Como canal de comunicação entre as bases de dados do Detran Estadual e Nacional com o aplicativo proposto neste trabalho, pensou-se na elaboração de um *Web Service*, que consiste em uma solução de integração de sistemas e comunicação entre diferentes aplicações.

Os *Web Services* são baseados nos padrões: XML (*Extensible Markup Language*), descreve e estrutura informações para a troca de mensagens; SOAP (*Simple Object Protocol*), protocolo responsável por vincular aplicações a serviços *Web*; UDDI (*Universal Description, Discover and Integratgion*), padrão para publicação ou localização de serviços *Web*; WSDL (*Web Services Description Language*), padrão que utiliza XML para descrever serviços *Web*; e HTTP (*HyperText Transfer Protocol*), protocolo responsável por realizar o transporte das mensagens, no formato XML entre as aplicações (HENDRICKS et. al., 2002).

Visto que o SOAP estabelece um vinculo entre sistemas e serviços *Web*, este foi utilizado para a troca de mensagens entre o *Web Service* desenvolvido e os sistemas do Detran, que possuem as informações necessárias para o aplicativo Android.

A estrutura deste trabalho é organizada da seguinte forma: o capítulo 2 (dois) apresenta conceitos relacionados à *Web Services*, introduzindo e conceituando seus conceitos, bem como abordar sobre sua arquitetura, modelo de comunicação e camadas. Em seguida, descrevem-se os conceitos relacionados ao protocolo de comunicação, abordando sobre o modelo de troca de mensagens SOAP, sua relação com XML, seus elementos, dentre outros. Por fim, apresenta-se os conceitos relevantes para este trabalho referente à plataforma Android, de forma a descrever sua arquitetura, os conceitos relacionados ao desenvolvimento, interface e publicação de aplicações para a esta plataforma.

O capítulo 3 (três) descreve os materiais utilizados para o desenvolvimento deste trabalho, além da metodologia adotada. Aborda-se ainda as instalações e configurações necessárias de ferramentas para a implementação de aplicações Android.

Já o capítulo 4 (quatro) relata os resultados obtidos na execução da proposta de desenvolvimento deste trabalho, discriminando todas as atividades desenvolvidas durante a concretização deste trabalho, atingindo-se assim o objetivo final proposto.

Por fim, o capítulo 5 (cinco) apresenta as conclusões obtidas durante todo o processo de desenvolvimento deste trabalho.

2 REFERENCIAL TEÓRICO

Este capítulo apresentará os principais conceitos e definições relacionados a *Web Services*, à SOAP (*Simple Object Access Protocol*) e ao Sistema Operacional Android. O entendimento destes conceitos é essencial para o desenvolvimento do aplicativo proposto neste trabalho, um aplicativo Android capaz de prover informações relacionadas a veículos gerenciados por sistemas de informação do Detran no País e aos condutores cadastrados efetivamente no Estado do Tocantins.

2.1. Web Services

Diversas tecnologias voltadas para o desenvolvimento de aplicações na Internet têm surgido nos últimos anos. Dentre essas novas tecnologias, a orientação a objetos é a que marcou o começo de um novo padrão de projeto e programação (ALBINADER & LINS, 2006, p. 5). Outra tecnologia que teve uma contribuição significativa para a interoperabilidade entre sistemas e aplicações heterogêneas foi a linguagem XML[1] (*Extensible Markup Language*) (ALBINADER & LINS, 2006, p. 5).

As aplicações disponíveis via Internet exigem um grande esforço de adequação quando se pretende que estas interajam entre si de forma a compartilhar informações. Tal esforço se refere às necessidades de adequações indispensáveis para estabelecer um relacionamento altamente interdependente e direcionado, uma vez que tais aplicações podem estar baseadas em diferentes linguagens de programação, plataformas de desenvolvimento e plataformas de *hardware*.

Com a necessidade de sistemas e aplicativos corporativos se comunicarem por meio da Internet a fim de estabelecer a troca de dados, surge o *Web Services*. Este tem como objetivo permitir a interação entre aplicações, conservando um baixo grau de acoplamento e pouco empenho de adaptação por meio de interfaces para acesso a informações disponíveis em XML (ALBINADER & LINS, 2006, p. 6).

Deste modo, segundo Albinader & Lins (2006, p. 11), *Web Service* pode ser definido por dois pontos de vista técnicos diferentes: a visão técnica e conceitual. Sendo assim, segundo o autor, tecnicamente *Web Service* compõe-se em um *software* de baixa interdependência, reutilizável, contendo componentes preparados para serem acessados via Internet. Pelo ponto de vista conceitual, o autor define *Web Services* como um serviço que disponibiliza uma interface padrão acessível pela Internet a fim de promover a interação entre programas distintos, sendo tal comunicação realizada por XML.

A partir do conceito apresentado anteriormente, conclui-se que *Web Services* são serviços que podem ser encontrados e disponibilizados na Internet para a comunicação entre sistemas diferentes, independente de que plataforma e/ou linguagem de programação estes sistemas foram desenvolvidos. De forma geral, *Web Services* permitem que as funcionalidades de um sistema sejam identificadas, localizadas e invocadas facilmente por outros componentes.

Os *Web Services* podem ser utilizados para diversos fins, tais como: mercado de ações, conversão de moeda, busca de artigos em um arquivamento, emissão de email, tradução entre idiomas, geração de um *Universally Unique Identifier* (UUID), informações de voo em tempo real, sistema de armazenamento de documentos XML, conversão entre vocabulários XML, variação de acesso a páginas *Web*, catálogos de endereços compartilhados, entre outros (DAUM, 2002). Enfim, *Web Services* podem ser utilizados em quaisquer aplicações que necessitem de integrar e conectar diferentes informações externas utilizando a Internet, sem a intervenção direta dos usuários.

2.1.1. Arquitetura de Web Services

Os serviços *Web* são construídos segundo um modelo genérico baseado na interação entre papeis e operações, denominado arquitetura e integração de *Web Services*. Neste modelo, os papeis são considerados categorias formadas por diferentes tipos de entidades existentes na aplicação a ser criada e as operações são funções fundamentais realizadas pelos papeis para a execução de um serviço (HENDRICKS et. al., 2002).

Os papéis podem ser providos de serviços, registro de serviços e consumidor de serviços. As operações básicas realizadas por estes papéis são: publicar, pesquisar e vincular. Na operação de publicação, o serviço é disponibilizado em um Registro de *Web Services* por meio de um provedor de serviços. Já na operação de pesquisa, é realizada uma consulta pelo consumidor de serviços no provedor de serviços. Por fim, a operação de vínculo é realizada

entre o consumidor e o provedor de serviços para acessar os serviços registrados. A Figura 1 apresenta a integração entre os elementos que compõe o modelo genérico de *Web Services*.

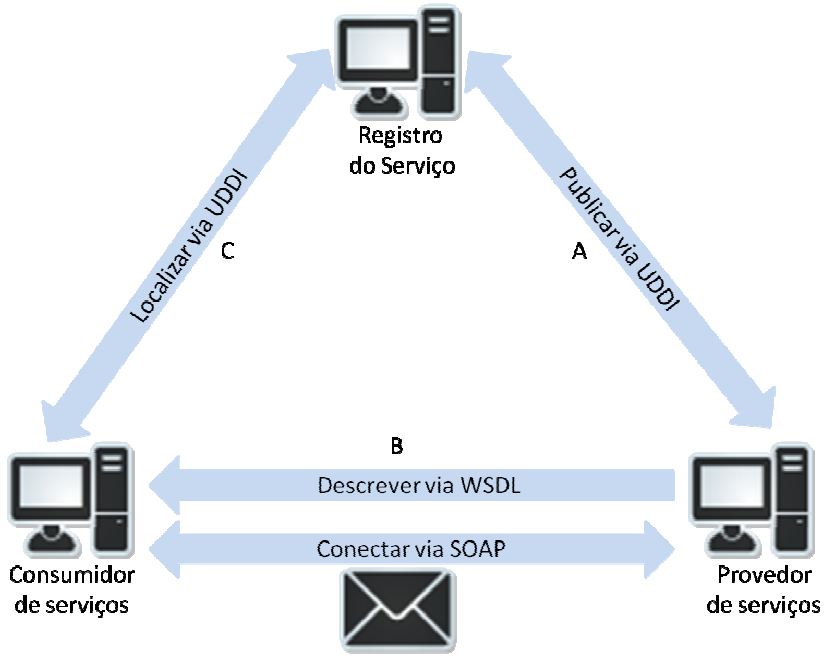

Figura 1: Modelo Genérico de um Web Services (modificado de CBDI Web Services Roadmap, 2005).

Como pode ser observado na Figura 1 parte A, o provedor de serviços é a plataforma acessada na solicitação do serviço, sendo responsável por descrever a mensagem em algum formato padrão. Como, por exemplo, XML e publicar os detalhes em um diretório central via UDDI (*Universal Description Discovery and Integration*), responsável por gerenciar informações sobre provedores, implementações e metadados de serviços. Tais detalhes envolvem uma descrição que detalha a interface do serviço que abrange suas operações e as mensagens de entrada e saída de cada operação. A partir disso, cria-se uma definição de como enviar cada mensagem para o endereço em que o *Web Service* está localizado, estabelecendo assim a forma pela qual será estabelecida a comunicação entre o serviço e as mensagens de retorno para cada operação. O registro do serviço é o local em que os provedores publicam as descrições dos serviços. A partir da descrição do serviço é possível descobrir em que local está um *Web Service* e como invocá-lo.

O consumidor de serviços consiste em uma aplicação que invoca ou inicializa uma interação com um serviço, sendo este serviço invocado tanto por um *browser* ou por outro *Web Service* (parte B da Figura 1). O consumidor de serviço encontra uma descrição de

serviço ou consulta o registro do serviço, obtendo assim informações de ligação da descrição do serviço durante a fase de desenvolvimento, que ocorre por meio de uma ligação estática, ou em tempo de execução, que caracteriza uma ligação dinâmica (parte C da Figura 1).

De forma geral, no modelo genérico de funcionamento de um *Web Service* o provedor publica um serviço na *Web* através de um registro UDDI (parte A da Figura 1), sendo pesquisado por um consumidor (parte C da Figura 1), que possui um vinculo com o provedor (parte B da Figura 1), possibilitando assim total acessibilidade aos serviços *Web* registrados no UDDI.

Sendo assim, Basiura et. al. (2002) afirma que, baseado no modelo genérico, o ciclo de vida de um *Web Service* restringe-se a seis itens: criação de um *Web Services*, divulgação do *Web Services* em um registro UDDI, localização do *Web Services*, obtenção da descrição do *Web Services* através da WSDL no momento do acesso do consumidor do *Web Services* a um provedor *Web Services*, criação do *Proxy* (servidor responsável por atender a requisições a fim de repassar os dados do cliente à frente) e do cliente e, por fim, chamada do *Web Services* através do SOAP. A próxima seção apresenta os principais conceitos relacionados ao modelo de comunicação *Web Services*.

2.1.2. Modelo de Comunicação

A arquitetura de um *Web Service* permite a implementação de dois modelos de comunicação, sendo eles: o baseado em RPC (*Remote Procedure Call*) síncrono ou o modelo de comunicação baseado em mensagens síncronas ou assíncronas (ALBINADER & LINS, 2006, p. 47). A Figura 2 representa o processo de comunicação baseado em RPC.

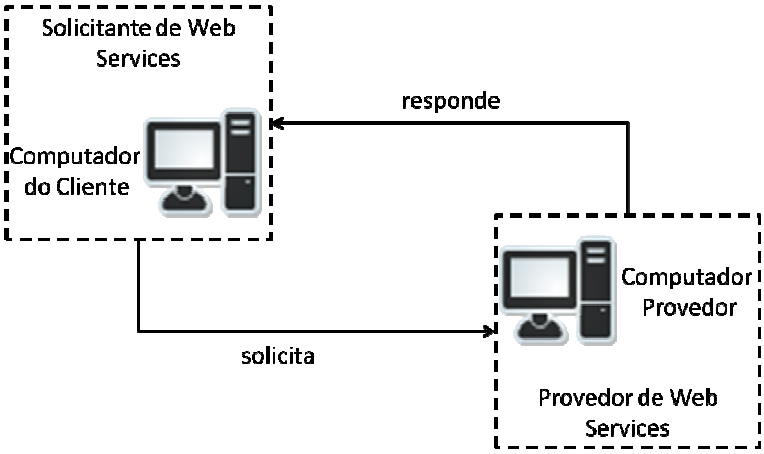

Figura 2: Modelo de comunicação baseado em RPC (modificado de ALBINADER & LINS, 2006, p. 48).

O modelo de comunicação baseado em RPC é caracterizado por possibilitar a troca de mensagens de maneira síncrona, em que há o par solicitação/resposta. Como pode ser observado na Figura 2, este modelo tem sua ideia iniciada a partir do momento que o cliente envia a solicitação, e é encerrado somente após o servidor enviar a resposta ao cliente. Enquanto o cliente não recebe a resposta do servidor, nenhuma outra operação é realizada. Sendo assim, pode-se dizer que este modelo apresenta alto grau de acoplamento. O cliente invoca os serviços disponíveis nos provedores de *Web Services* e, eventualmente, podem enviar parâmetros com valores para que estes serviços executem determinadas operações. Uma vez que estes serviços são executados, são retornados aos clientes resultados que podem ser tanto valores atômicos quanto valores complexos como, por exemplo, objetos (ALBINADER & LINS, 2006, p. 48).

O outro modelo de comunicação de *Web Services* é baseado em mensagens, o qual está representado na Figura 3.

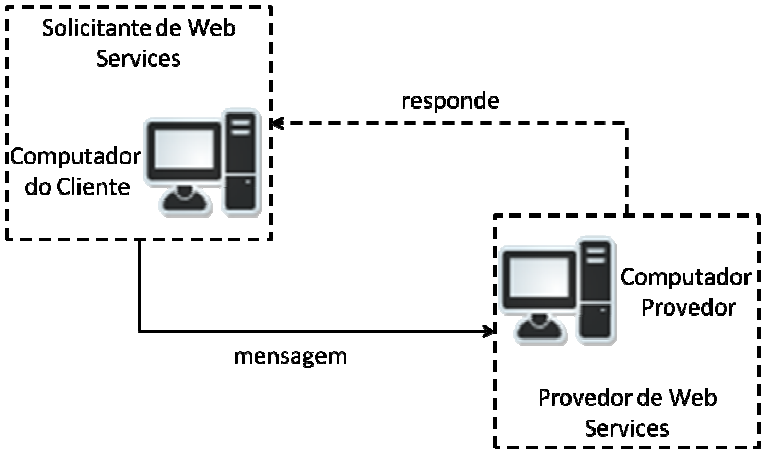

Figura 3: Modelo de comunicação baseado em mensagens (modificado de ALBINADER & LINS, 2006, p. 49).

O modelo de comunicação baseado em mensagens é focado na comunicação dirigido a documento, definindo baixo acoplamento (ALBINADER & LINS, 2006, p. 49). Como pode ser observado na Figura 3, este modelo tem sua ideia iniciada a partir do momento que o cliente invoca algum serviço disponibilizado pelo provedor dos *Web Services*. No momento que o serviço é invocado, neste modelo de comunicação baseado em mensagens, o servidor recebe um documento completo com as informações necessárias para a execução do método. Neste ponto percebe-se uma diferença entre o outro modelo apresentado anteriormente, o qual recebe um conjunto de parâmetros para realizar tal execução. Quando o servidor recebe o documento enviado pelo cliente, o servidor processa este documento, enviando ou não uma resposta de volta ao cliente.

Ao implementar o modelo de comunicação baseado em mensagens, pode-se definir que o cliente envie ou receba documentos de forma assíncrona de e para um *Web Service* (ALBINADER & LINS, 2006, p. 49). Contudo, vale ressaltar que estas operações não podem ser realizadas no cliente de forma simultânea. Sendo assim, conclui-se que quando o cliente envia a solicitação e aguarda até obter uma resposta, tem-se o modelo de comunicação baseado em mensagem síncrona. Ao contrário, quando o cliente envia uma solicitação sem a perspectiva de receber uma resposta, tem-se o modelo de comunicação baseado em mensagem assíncrona.

Fazendo uma analogia sobre os dois modelos de comunicações apresentados nesta seção, notam-se consideráveis diferenças entre os modelos, os quais são listados na Tabela 1.

Tabela 1 – Paralelo entre os modelos de comunicações baseados em RPC e mensagens.

Modelo de comunicação baseado em RPC	Modelo de comunicação baseado em mensagens
É caracterizado pelo par solicitação/resposta.	Pode ou não receber uma resposta.
Comunicação baseada em mensagem síncrona.	Comunicação baseada em mensagem síncrona ou assíncrona.
Alto grau de acoplamento.	Baixo grau de acoplamento.
Há a passagem de parâmetros para as chamadas de métodos para os provedores de *Web Services*.	Há a passagem de um documento completo para as chamadas de métodos para os provedores de *Web Services*.
Tipo de retorno: valores atômicos ou complexos.	Tipo de retorno: documento.

O modelo de comunicação é definido de acordo com as necessidades do sistema a ser implementado. Um exemplo de sistema que possui seu modelo de comunicação baseado em mensagem pode ser observado em aplicações destinadas a comunicação entre usuários, como visto no envio de mensagens entre celulares através de operadoras telefônicas, entretanto, utilizando a comunicação através da Internet. O envio da mensagem pode ou não obter um retorno, ficando a critério do usuário estipular um *feedback* para a mensagem enviada. Esta transmissão da mensagem pode ser definida de forma síncrona ou assíncrona, de acordo com as especificações da aplicação. O envio de informações seria através de documentos que contenham a mensagem a ser enviada e, caso for necessário um retorno, um documento também seria enviado ao remetente contendo o *feedback* solicitado.

A aplicação apresentada neste trabalho tem sua estrutura de comunicação baseada no modelo RPC pela necessidade obrigatória de enviar mensagem e receber resposta, sendo seu modelo de comunicação baseado no método síncrono. A solicitação dos serviços é baseada na passagem de parâmetros e, com base nestes, terá a resposta solicitada através de uma estrutura complexa de dados.

Tendo apresentado os dois modelos de comunicação, a próxima seção apresentará os aspectos fundamentais dos elementos que compõem a tecnologia de *Web Services*.

2.1.3. Camadas dos Web Services

Os *Web Services* são baseados em protocolos e linguagens padrão da *Web*, definidos pela W3C, que são: *Universal Description Discovery and Integration* (UDDI), *Web Service Description Language* (WSDL), *Simple Object Access Protocol* (SOAP), *Extensible Markup Language* (XML) e *HyperText Transfer Protocol* (HTTP). A Figura 4 representa as camadas mencionadas.

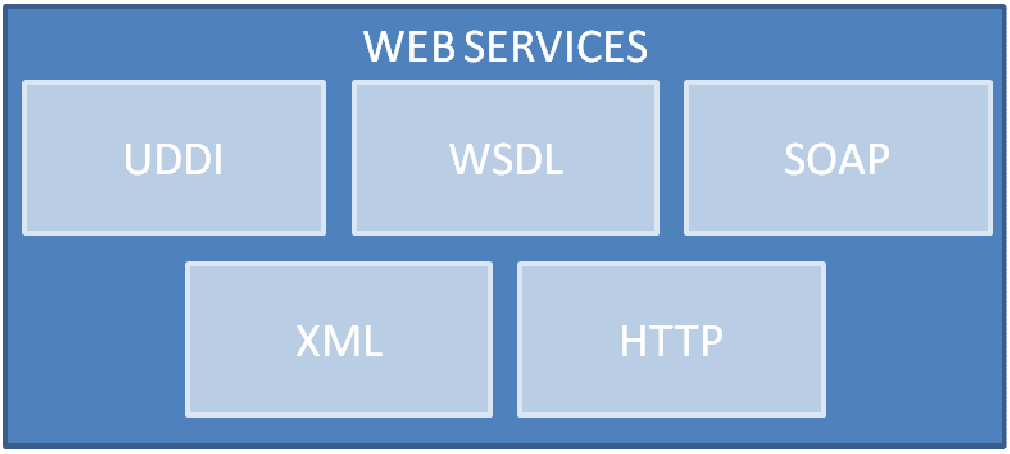

Figura 4: Camadas dos Web Services.

Todos esses elementos que compõem a tecnologia de *Web Services*, representados na Figura 4, serão apresentados nas próximas seções.

2.1.3.1. Diretório para Web Services

O *Universal Description Discovery and Integration* (UDDI) foi criado pela indústria de desenvolvimento de *software* e consiste em um padrão que possibilita a descrição, publicação e busca de informações sobre os *Web Services* oferecidos na Internet (UDDI, 2002). O desenvolvimento deste diretório iniciou com o objetivo de facilitar a localização de serviços *Web* disponíveis, de um modo aberto, padronizado e independente de plataforma (ALBINADER & LINS, 2006, p. 36).

Anteriormente à criação do UDDI, a indústria não possuía um método padronizado de ofertar maiores informações relacionadas a produtos e *Web Services* disponibilizados aos clientes e parceiros. Não havia também uma definição constante e padrão quanto a forma de integração dos sistemas e processos já constituídos e disponíveis, bem como a forma que tal

integração seria realizada entre os parceiros de negócios que chegassem a se interessar pelos *Web Services* ofertados (JEWELL & CHAPPELL, 2012, online, tradução nossa).

Segundo Albinader & Lins (2006, p. 37), o registro UDDI fornece três formas de expor um negócio: páginas brancas, páginas amarelas e páginas verdes. Tais formas serão apresentadas abaixo, conforme Albinader & Lins (2006, p. 37):

- páginas brancas: possuem informações relacionadas a contato e identificadores da empresa, que envolvem o nome do negócio, endereço, regras de tributação em que se classifica e dados relativos a cadastros oficiais. A partir das informações contidas neste tipo de página, *Web Services* podem ser encontrados por meio da identificação de cada empresa;
- páginas amarelas: descrevem *Web Services* separando-os em categorias. Tais categorias abrangem grupos de *Web Services* que apresentam similaridades entre si;
- páginas verdes: possuem informações técnicas relacionadas ao comportamento e funções toleradas pelos *Web Services* disponibilizados pelas empresas. Tais informações envolvem apontadores para grupos exclusivos de informações mais especializadas, bem como indicam o local que os *Web Services* podem ser encontrados.

A utilização do UDDI se justifica em um ambiente em que há diversos *Web Services* disponíveis para escolha e utilização. Sob o ponto de vista das empresas consumidoras de *Web Services*, Albinader & Lins (2006, p. 38) afirma que um analista de negócios deve buscar um ou mais UDDI por *Web Services* que suprem suas condições e particularizações. Para isso, existem diversos sites especializados em negócios e sites de mercado que permitem a comparação entre *Web Services* análogos disponíveis em UDDIs diferentes por meio da extração e tratamento de informações mais apropriadas para a tomada de decisão e analise humana (JEWELL & CHAPPELL, 2012, online, tradução nossa). Já sob o ponto de vista dos desenvolvedores de *Web Services*, Albinader & Lins (2006, p. 38) afirma que as integrações das UDDIs podem ser feitas através de APIs (*Application Programming Interfaces*), as quais possibilitam que seja feita a publicação e consulta nas UDDIs.

O segundo elemento que compõe a tecnologia de *Web Service*, *Web Service Description Language* (WSDL), será apresentado na próxima seção.

2.1.3.2. Linguagem para Descrição de Web Services

O *Web Service Description Language* (WSDL) é uma linguagem de marcação que descreve um *Web Service*, informando dados quanto ao serviço que ele oferece como, por exemplo, a forma que a comunicação é realizada e onde está localizado. Ainda, disponibiliza um mecanismo estruturado para detalhar as operações que podem executar, o formato das mensagens que processa, os protocolos que suporta e o ponto de acesso de uma instância de um *Web Service*. De forma resumida, um *Web Service* expõe sua interface para os usuários através do WSDL, que pode se comunicar com o *Web Service* direta ou indiretamente.

Com a utilização de um WSDL, é possível encontrar quais são os tipos de dados, formatos de mensagens e serviços disponibilizados por um *Web Service* (SAMPAIO, 2006, p. 38) – os tipos de dados variam conforme os serviços disponíveis. Também é exibida a forma de retorno deste *Web Service*, que pode ser estruturada em XML, contendo os campos de retorno do serviço.

O uso da WSDL é primordial, uma vez que sem ela não seria possível conhecer os métodos de um *Web Service*, impossibilitando assim sua utilização. É no WSDL que o formato da mensagem é descrito através de uma XML *Schema*, por ser baseado em padrões bem definidos e não depender de uma linguagem de programação, tornando-o desta forma adequado para descrever as interfaces dos *Web Services* (FILHO & ZARA, 2002, p. 72). Destaca-se que uma XML *Schema* consiste em uma linguagem de esquema baseada em XML. Esta linguagem tem como finalidade definir regras de validação em documentos no formato XML.

A Listagem 1 apresenta um exemplo de um documento WSDL simples, que visa fornecer as cotações de ações. Este documento tem como entrada um símbolo do tipo *string* e retorna o preço do tipo *float*.

```xml
<?xml version="1.0"?>
<definitions name="StockQuote" targetNamespace="http://example.com/stockquote.wsdl"
             xmlns:tns="http://example.com/stockquote.wsdl"
             xmlns:xsd1="http://example.com/stockquote.xsd"
             xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
             xmlns="http://schemas.xmlsoap.org/wsdl/">

    <types>
       <schema targetNamespace="http://example.com/stockquote.xsd"
        xmlns="http://www.w3.org/2000/10/XMLSchema">
          <element name="TradePriceRequest">
            <complexType>
                <all>
                    <element name="tickerSymbol" type="string"/>
                </all>
            </complexType>
          </element>
          <element name="TradePrice">
            <complexType>
                <all>
                    <element name="price" type="float"/>
                </all>
            </complexType>
          </element>
       </schema>
    </types>

    <message name="GetLastTradePriceInput">
        <part name="body" element="xsd1:TradePriceRequest"/>
    </message>

    <message name="GetLastTradePriceOutput">
        <part name="body" element="xsd1:TradePrice"/>
    </message>

    <portType name="StockQuotePortType">
        <operation name="GetLastTradePrice">
          <input message="tns:GetLastTradePriceInput"/>
          <output message="tns:GetLastTradePriceOutput"/>
        </operation>
    </portType>

    <binding name="StockQuoteSoapBinding" type="tns:StockQuotePortType">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="GetLastTradePrice">
          <soap:operation soapAction="http://example.com/GetLastTradePrice"/>
          <input>
              <soap:body use="literal"/>
          </input>
          <output>
              <soap:body use="literal"/>
          </output>
        </operation>
    </binding>

    <service name="StockQuoteService">
        <documentation>My first service</documentation>
        <port name="StockQuotePort" binding="tns:StockQuoteBinding">
          <soap:address location="http://example.com/stockquote"/>
        </port>
    </service>

</definitions>
```

Listagem 1: Exemplo de documento WSDL (W3C, 2001, online).

Como pode ser observado na Listagem 1, um documento WSDL pode ser formado pelos seguintes elementos: *definitions*, *types*, *message*, *import*, *operation* e *portType*, *binding*, *service* e *port*. O primeiro elemento a compor um documento WSDL é o <definitions>, sendo desta forma a raiz de qualquer documento WSDL, a qual define os atributos que definem os *namespaces* que serão utilizados no documento. Este elemento está definido entre as linhas 2 e 6 da Listagem 1.

Entre as linhas 8 e 26 da Listagem 1, definiu-se o elemento <types>, que representa os tipos de dados que estão presentes na mensagem. Já entre as linhas 28, tem-se a definição dos dados que serão transmitidos, representada pelo elemento <message>. O conteúdo da mensagem representando os parâmetros que são passados e a resposta que o serviço retorna consta no elemento <part>, definido nas linhas 29 e 33.

O elemento <porType> definido nas linhas 36 e 41 da Listagem 1 possui um conjunto de operações (<operation>, definido nas linhas 37 e 40), que definem o parâmetro utilizado na operação. O elemento <input> definido na linha 38 representa o parâmetro de entrada da operação (utilizado na chamada ao método) enquanto que o elemento <output> definido na linha 39 representa o parâmetro de saída da operação (resposta retornada pelo método).

As linhas 43 e 54 da Listagem 1 define o espaço do elemento <binding>, responsável por mapear o elemento <operation> em um elemento <portType> a fim de definir um protocolo específico, definido entre as linhas 45 e 53. No exemplo em questão, o elemento <binding> associa o elemento <portType> ao protocolo SOAP, como pode ser observado nas linhas 44 à 53.

Por fim, define-se a localização real do serviço por meio dos elementos <service> e <port> (linhas 56 à 61 da Listagem 1). É necessário que seja especificada uma porta exclusiva para cada tipo de ligação descrita no elemento <binding>.

De forma geral, segundo Albinader & Lins (2006, p. 36), a principal função do WSDL é realizar a integração entre o fornecedor do serviço e o consumidor, de forma que ambos tenham acesso ao WSDL e possam se comunicar a fim de trocar informações em um formato previamente definido. Vale ressaltar que nesta comunicação não há a preocupação de informações quanto ao sistema operacional ou plataforma que os participantes estejam utilizando.

O terceiro elemento que compõe a tecnologia de *Web Service*, o *Single Object Access Protocol* (SOAP), será apresentado na próxima seção.

2.1.3.3. Protocolo de Comunicação para Web Services

O *Single Object Access Protocol* (SOAP) consiste em um protocolo responsável por invocar aplicações remotas por meio de RPC ou troca de mensagens em ambientes independente de plataforma e linguagem de programação. Este protocolo é utilizado para toda e qualquer comunicação e utilização de *Web Services*, o qual realiza o transporte de conteúdos entre *Web Services* e seus programas clientes.

A comunicação acontece entre duas aplicações de diferentes plataformas, as quais podem se comunicar sem a necessidade de conhecimento da arquitetura de ambas as partes, somente realizando a solicitação do serviço e recebendo o seu retorno, sem necessariamente ter o conhecimento do processamento executado pelos *Web Services*.

A mensagem transportada pelo SOAP na comunicação com *Web Services* é baseada em XML. O conteúdo desta mensagem não possui seu significado semântico completo. Devido a isso, deve-se definir quais informações cada mensagem deve transportar, bem como a sua relevância para o *Web Service* como um todo (ALBINADER & LINS, 2006, p. 35). Em alguns casos, há a necessidade ainda das mensagens SOAP conterem informações quanto a parâmetros necessários para a execução de métodos, sendo tais métodos invocados dentro do mesmo uso do *Web Service* em questão (ALBINADER & LINS, 2006, p. 35).

De forma geral, a utilização do SOAP na implementação de um *Web Service* garante a interoperabilidade e intercomunicação entre sistemas distintos a partir de uma linguagem de programação e mecanismo de transporte padrões, sendo eles o XML e HTTP, respectivamente. A seção 2.2 apresentará este conceito de forma mais detalhada.

O quarto elemento que compõe a tecnologia de *Web Service*, o *Extensible Markup Language* (XML), será apresentado na próxima seção.

2.1.3.4. Linguagem de Marcação para Web Services

A *Extensible Markup Language* (XML) é um padrão para formato de dados na Internet que possibilita descrever, armazenar, intercambiar e manipular dados estruturados (XML, 2005). Por ser considerada uma linguagem de marcação de dados, a XML facilita declarações precisas de conteúdo e resultados, além de possibilitar a geração de novas aplicações de manipulação e visualização de dados através da *Web*.

Devido a esta facilidade, XML tornou-se o padrão para o transporte de dados estruturados, conteúdos e formato para dados que representam documentos em meio eletrônico (ALBINADER & LINS, 2006, p. 41).

A XML, no contexto da tecnologia de *Web Services*, apresenta as vantagens de ser prontamente transportável, compatível e comum com a camada de rede pela qual é conduzida em todos os formatos de comunicação.

O último elemento que compõe a tecnologia de *Web Service*, o *Hypertext Transfer Protocol* (HTTP), será apresentado na próxima seção.

2.1.3.5. Protocolo de Tranporte para Web Services

O *Hypertext Transfer Protocol* (HTTP) é um padrão de Internet que consiste em um protocolo capaz de possibilitar a troca de dados pela *Web* entre servidores *Web* e *browsers*. Assim, a principal funcionalidade do HTTP é realizar a comunicação entre servidores *Web* (aplicações servidoras de páginas) e *browsers* (leitores de páginas) e, consequentemente, por todo o ambiente da Internet.

A comunicação realizada pelo HTTP é estabelecida por meio de conexões TCP (*Transmission Control Protocol*) confiáveis através da porta 80. Sendo assim, é por meio da utilização do HTTP que a confiabilidade de *Web Services* é garantida (ALBINADER & LINS, 2006, p. 41).

Conforme foi apresentado, as categorias no modelo de *Web Services* são responsáveis pela geração de operações funcionais de pesquisa de serviço *Web*, seu vinculo e sua publicação, sendo que existem padrões que devem ser adotados para cada operação. Dentre os padrões a serem adotados, citados anteriormente, encontram-se: UDDI, WSDL, SOAP, XML e HTTP. A seção 2.2 apresenta uma análise geral do padrão SOAP, uma vez que este foi utilizado no desenvolvimento do aplicativo proposto para estabelecer a chamada a procedimentos e respostas na forma de elementos XML na comunicação com os *Web Services* da aplicação.

2.2. Simple Object Access Protocol (SOAP)

SOAP (*Simple Object Access Protocol*) é um protocolo baseado em XML. Este protocolo permite a comunicação entre sistemas distribuídos estabelecidos sobre conjuntos de *software* e *hardware* heterogêneos, o qual foi projetado em um contexto em que as possibilidades para

sistemas distribuídos eram caracterizados por soluções baseadas em forte acoplamento entre as partes (ALBINADER & LINS, 2006, p. 78). Em suma, o SOAP permite que métodos remotos que proveem serviços sejam chamados por meio de uma conexão estabelecida pela aplicação-cliente.

O SOAP não define a semântica de uma aplicação como um modelo de programação ou a semântica de execução desta aplicação, mas define um mecanismo simples para expressar a semântica da aplicação. Devido a isso, ao criar um *Web Service* deve ser determinado que informações cada mensagem deverá carregar e qual a sua importância para o *Web Service* a ser publicado.

A utilização do XML na codificação de mensagens SOAP apresenta as seguintes vantagens (LEOPOLDO, 2012, p. 1):

- XML é uma linguagem fácil para entender e encontrar erros, por ser facilmente lida por seres humanos;
- XML *parsers* e tecnologias correspondentes são mundialmente disponíveis;
- XML é um padrão aberto;
- XML abrange tecnologias que podem fortalecer o SOAP;
- XML possui uma especificação simplificada.

A próxima seção apresenta o modelo de troca de mensagens SOAP.

2.2.1. O Modelo de Troca de Mensagens SOAP

As mensagens SOAP são transferidas seguindo um modelo emissor para receptor, sendo assim unidirecionais, em que há o processo de solicitação/resposta (W3C, 2000, p. 3, tradução nossa). O autor cita ainda que uma aplicação para receber uma mensagem SOAP deve seguir os seguintes procedimentos em ordem:

- verificar todo o conjunto da mensagem destinada à aplicação;
- verificar todas as partes obrigatórias identificadas na mensagem, validando seu suporte na aplicação e processá-la em conformidade. Caso não seja o caso, a mensagem deve ser rejeitada;
- caso o aplicativo SOAP não seja o destino final da mensagem, todo o conjunto da mensagem identificado no primeiro passo deve ser removido antes seja encaminhado.

Tanto o processamento de uma mensagem quanto de parte dela demanda que o processador SOAP compreenda o padrão de troca, o papel do receptor, o emprego dos

mecanismos RPC quando houver, a representação ou codificação de dados, além da semântica para um correto processamento (W3C, 2000, p. 4, tradução nossa). A próxima seção apresenta a estrutura das mensagens SOAP.

2.2.2. SOAP e XML

Toda mensagem SOAP é codificada por meio de XML. Todos os elementos atribuídos definidos no SOAP para a mensagem que ele gera deve incluir *namespace* adequado. Assim como também a aplicação deve ser capaz de processar *namespace* SOAP nas mensagens recebidas. Vale ressaltar que mensagens SOAP sem *namespaces* podem ser processadas e *namespaces* incorretos fazem com que as mensagens sejam descartadas. Segundo W3C (2000, p. 4, tradução nossa), SOAP define dois *namespaces*, sendo eles: http://schemas.xmlsoap.org/soap/envelope/ e http://schemas.xmlsoap.org/soap/encoding/.

Uma mensagem SOAP é composta por atributos e valores apenas, não podendo desta forma conter declaração de tipo de documentos e instruções de processamento. O formato XML para as mensagens trocadas entre o cliente e o *Web Service* é definido através do SOAP, permitindo assim que o cliente receba os dados e os manipule (FILHO & ZARA, 2002, p. 71), sendo esta comunicação estabelecida via HTTP (BECKER, CLARO & SOBRAL, 2002, p. 8).

A Figura 5**Erro! Fonte de referência não encontrada.** apresenta como é realizado o envio de mensagens SOAP/HTTP.

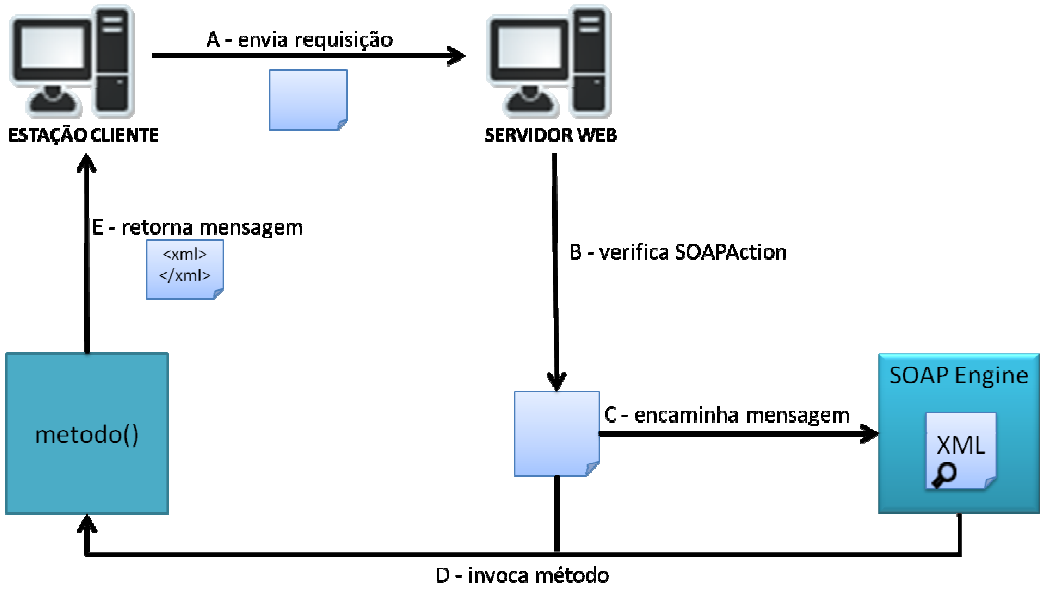

Figura 5: Envio de mensagens SOAP/HTTP.

Como apresentado na Figura 5**Erro! Fonte de referência não encontrada.**, o envio de mensagens SOAP/HTTP é realizado a partir de uma aplicação que está em um servidor *Web*. No momento em que a aplicação recebe uma requisição (Figura 5, parte A), o servidor *Web* verifica se existe o campo *SOAPAction* na requisição (Figura 5, parte B). Caso exista, a requisição é encaminhada para a SOAP *Engine* (Figura 5, parte C), que processa a parte XML da mensagem e executa uma "varredura" no registro local do endereço do serviço especificado, o qual possui informações referentes aos serviços disponíveis no servidor. Em seguida, o método especificado no serviço é localizado e invocado por reflexão (Figura 5, parte D). Por fim, a resposta SOAP é retornada ao cliente em um documento XML estruturado como em uma requisição (Figura 5, parte E). Este tipo de retorno só não é realizado quando o próprio método já possui um tipo de retorno contendo o resultado esperado pelo cliente, ou seja, caso na implementação do método seja definido um valor de retorno como, por exemplo, o valor do resultado obtido por um cálculo matemático, não cabe a utilização de uma mensagem XML para o retorno do resultado ao SOAP. A próxima seção apresentará de forma detalhada todos os elementos que compõem uma mensagem SOAP, conceituando cada um dos elementos.

2.2.3. Especificação dos elementos SOAP

Segundo W3C (2000, p. 2, tradução nossa) e Becker, Claro & Sobral (2002, p. 8), a mensagem SOAP deve ser codificada via XML e conter três elementos: SOAP Envelope, SOAP Header e SOAP Body, como apresentado na Figura 6**Erro! Fonte de referência não encontrada.**.

Figura 6: Estrutura do SOAP.

Como apresentado na Figura 6, a estrutura SOAP é composta por três elementos, sendo eles envelope, header e body, os quais serão detalhados a seguir.

- o SOAP Envelope define o conteúdo da mensagem e os vários *namespaces* que são usados pelo restante da mensagem, sendo o primeiro elemento do documento XML representando a mensagem;
- o SOAP Header define informação relacionada à autenticação, transação e contabilização;
- o SOAP Body define informações relacionadas aos métodos e parâmetros que serão invocados ou respostas retornadas. Entretanto, quando o SOAP é utilizado como RPC, o Body é composto por um único elemento que corresponde ao nome do método, parâmetros e a identificação do serviço.

Destaca-se que o Envelope e o Body são elementos obrigatórios em um documento XML, sendo apenas o Header um elemento opcional. Embora os elementos do SOAP sejam descritos em conjunto, o envelope e as regras de codificação são definidos em espaços diferentes a fim de promover a simplicidade através de módulos (W3C, 2000, p. 2, tradução nossa).

Segundo W3C (2000, p. 4, tradução nossa), as regras gramaticais do SOAP Envelope de uma mensagem são:

- O nome do elemento é “Envelope”;

- O elemento deve estar presente em uma mensagem SOAP;
- O elemento pode conter declarações de *namespace*, bem como atributos adicionais e elementos secundários (no caso da existência de atributos adicionais).

Já as regras gramaticais do SOAP Header de uma mensagem são (W3C, 2000, p. 5, tradução nossa):

- O nome do elemento é “Header”;
- O elemento pode estar presente em uma mensagem SOAP. Caso esteja presente, este deve ser o primeiro elemento filho logo abaixo de um elemento SOAP Envelope;
- O elemento pode conter um conjunto de entradas de cabeçalho, devendo ser cada entrada iniciada logo após o elemento filho do elemento SOAP Header.

Por fim, as regras gramaticais do SOAP Body de uma mensagem são (W3C, 2000, p. 5, tradução nossa):

- O nome do elemento é “Body”;
- O elemento deve estar presente em uma mensagem SOAP e deve estar logo após o elemento filho de um elemento SOAP Envelope. Entretanto, caso a mensagem seja composta pelo elemento Header, o Body deve estar logo abaixo do SOAP Header;
- O elemento pode conter um conjunto de entradas Body, sendo cada uma das entradas iniciadas imediatamente após o elemento filho de um elemento SOAP Body. SOAP define ainda o elemento de falha, utilizado para indicar as mensagens de erro.

2.2.4. Especificação do SOAP Envelope

O encapsulamento de dados a serem transferidos é definido por um conjunto de regras especificas pelo SOAP Envelope XML. Tais dados específicos, envolvem dados da aplicação desde nomes e parâmetros de métodos a serem chamados a valores que tais métodos retornam. Informações referentes a quem deve processar os conteúdos de Envelopes e a forma como implementar mensagens de erro no caso de falhas também são informações fornecidas na mensagem do SOAP Envelope. A próxima seção apresenta o segundo elemento, SOAP Header.

2.2.5. Especificação do SOAP Header

SOAP fornece um mecanismo flexível para estender uma mensagem de um modo descentralizado e modular, com a ausência de um prévio conhecimento entre as partes comunicantes (W3C, 2000, p. 6, tradução nossa). Exemplos típicos de extensões que podem ser implementadas como entradas de Header são: autenticação, gerenciamento de transações, pagamento, dentre outros.

O elemento Header é codificado como o primeiro elemento filho do elemento SOAP Envelope. Todos os elementos filhos imediatos ao elemento Header são chamados de entradas de cabeçalho.

W3C (2000, p. 6, tradução nossa) levanta que as regras de codificação das entradas de cabeçalho são:

- a entrada de cabeçalho é identificada por meio do seu nome de elemento qualificado, que consiste na URI (*Unique Resource Identifier*) do *namespace* e o nome local;
- o atributo SOAP *encodingStyle* pode ser utilizado para indicar o estilo de codificação utilizado para as entradas de cabeçalho;
- o atributo SOAP *mustUnderstand* e o atributo SOAP ator podem ser utilizados para indicar como e por quem executar a entrada.

A próxima seção apresenta o último elemento SOAP, o Body.

2.2.6. Especificação do SOAP Body

W3C (2000, p. 7, tradução nossa) afirma que o elemento SOAP Body fornece um mecanismo simples para a troca de informação obrigatória designada para o destinatário final da mensagem. O autor afirma ainda que as utilizações mais comuns do elemento Body incluem chamadas de triagem RPC e relatórios de erros. Como já mencionado o elemento SOAP Body é codificado em um elemento filho imediato ao elemento SOAP Envelope XML. Caso haja o elemento SOAP Header, o Body deve ser o primeiro elemento filho imediato ao elemento Envelope.

Todos os elementos filhos imediatos ao elemento Body são conhecidos como entradas do corpo e cada entrada é codificada como um elemento independente dentro do elemento SOAP Body.

As regras de codificação das entradas do corpo são (W3C, p. 7, tradução nossa):

- A entrada do corpo é identificada pelo seu nome de elemento totalmente qualificado, que consiste no *namespace* URI e o local.

- O atributo SOAP *encondingStyle* pode ser utilizado para indicar o estilo utilizado para as entradas do corpo.

Enquanto o Header e o Body são definidos como elementos independentes, de fato, eles são relacionados. Isso porque uma entrada de corpo é semanticamente equivalente a uma entrada de cabeçalho destinado ao ator padrão e com um atributo SOAP *mustUnderstand* com valor “1”. O ator padrão é indicado por não utilizar o atributo ator.

SOAP define uma entrada do corpo, que é a entrada de falha utilizada para relatar erros, a qual será apresentada na próxima seção.

2.2.7. Especificação do SOAP Fault

O elemento SOAP Fault é utilizado para transportar erro e/ou informação de *status* em uma mensagem SOAP (W3C, 2000, p. 7, tradução nossa). Quando o elemento é utilizado, este deve aparecer como uma entrada do corpo e não pode aparecer mais de uma vez dentro de um elemento SOAP Body.

Conforme W3C (2000, p. 7, tradução nossa), o elemento SOAP Fault define quatro subelementos:

- Faultcode – este elemento é utilizado por *software* a fim de fornecer um mecanismo que identifica falhas. O Faultcode deve estar presente em um elemento SOAP Fault. SOAP define um conjunto de códigos de falha SOAP a fim de cobrir falhas SOAP básicas. O elemento Faultcode visa fornecer uma explicação legível da falha e não se destina ao processamento do algoritmo. Ele deve estar presente em um elemento de falha SOAP e deve fornecer ao menos informações explicando a origem da falha.
- Faultactor – este elemento objetiva fornecer informações sobre quem causou a falha ocorrida no percurso da mensagem, indicando a fonte da falha. O valor do atributo Faultactor é um URI de identificação da fonte. Os aplicativos que não funcionam como o melhor destino da mensagem SOAP devem incluir no elemento SOAP Fault o subelemento Faultactor. O destino final de uma mensagem pode utilizar o Faultactor para indicar explicitamente o que gerou o erro.
- Detail – este elemento é utilizado no transporte de informações especificas de erros de aplicações, sendo tais erros relacionados ao elemento Body. O Detail deve ser utilizado quando o conteúdo do elemento Body correr o risco de ser processado sem êxito. Vale ressaltar que o elemento em questão não deve transportar informações sobre entradas de

cabeçalho. Sendo assim, o transporte de informações detalhadas referente a entradas de cabeçalho deve ser realizado dentro das entradas de cabeçalho. A ausência do elemento Detail no elemento Fault indica que a falha não está relacionada ao processamento do elemento Body.

Abordando ainda mais sobre o elemento Detail, todos os elementos filhos imediatos ao mesmo são chamados de entradas de detalhes e cada entrada de detalhe é codificada como um elemento independente dentro do elemento Detail.

As regras de codificação das entradas de detalhes são (W3C, 2000, p. 8, tradução nossa):

- A entrada de detalhes é identificada por seu nome de elemento totalmente qualificado, que consiste no elemento URI e o local. Elementos filhos imediatos ao elemento Detail pode ser qualificado de *namespace*;
- O atributo SOAP *encodingStyle* pode ser utilizado para indicar o estilo de codificação utilizada para as entradas de detalhe.

Segundo Rosenberg (2004, apud SILVA & CUNHA, p. 18), as mensagens SOAP devem ter seu conteúdo conservado em sigilo dos intermediários, quando não-autorizados, até chegarem aos seus destinos, por questão de segurança. Ainda segundo o autor, assim que a mensagem chega ao destinatário, este deve reconhecer quem enviou os dados nela contidos e quais acessos o emissor tem direito.

2.3. Plataforma Android

Android consiste em um conjunto de *softwares* como sistema operacional, *middleware* e aplicações chave para dispositivos móveis. O sistema operacional de um Android é responsável por proporcionar navegação na Internet, captura de imagens e vídeos, criação e visualização de documentos, anotações de calendário, gerenciamento de contatos, localização por GPS e navegação por mapas (SIMÃO et. al., 2011, p. 1).

Ferramentas e APIs (*Application Programming Interfaces*) para o desenvolvimento de aplicativos Android são fornecidos na versão Beta do Android SDK, sendo a linguagem de programação utilizada Java. Esta API oferece os seguintes recursos (ANDROID SDK, 2012):

- *framework* de desenvolvimento de aplicação: possibilita a reutilização e a substituição de componentes, além de facilitar o acesso a recursos exclusivos e manutenção;

- Dalvik: máquina virtual criada e otimizada para a utilização em dispositivos móveis e suas restrições;
- navegador *web* integrado: é integrado e baseado no WebKit engine, sendo seu código aberto;
- biblioteca de gráficos: disponibiliza por meio de biblioteca gráficos otimizados e personalizados para dispositivos móveis, estes gráficos podem ser de duas e três dimensões, com aceleração de *hardware* opcionalmente;
- SQLite: armazenamento de dados estruturados;
- suporte para mídia: disponibiliza a execução de áudio, vídeo e imagem estática em diversos formatos como MPEG4, H.264, MP3, AAC, AMR, JPG, PNG, GIF;
- telefonia com tecnologia GSM: permite a manipulação de operações telefônicas, dependendo de acesso por parte do fabricante;
- *bluetooth*, EDGE, 3G e *WiFi:* disponibiliza tecnologias de transmissão de dados sem fio, mas depende da concessão do fabricante;
- câmera, GPS e bússola: fornece interação com redes sociais, contudo depende da permissão do fabricante;
- ambiente de desenvolvimento com *plugin* para a IDE Eclipse: oferece dispositivo emulador, ferramentas de depuração, supervisão de memória e desempenho.

A próxima seção apresentará a arquitetura da plataforma para o desenvolvimento de aplicativos para dispositivos móveis baseados em Android.

2.3.1. Arquitetura da plataforma Android

A arquitetura da plataforma Android é composta por quatro camadas, sendo elas: Kernel GNU Linux, bibliotecas, *framework* para aplicações e as próprias aplicações. Há ainda a camada *runtime*, responsável por executar os aplicativos do dispositivo. A Figura 7 apresenta a arquitetura da plataforma Android.

Figura 7: Arquitetura da Plataforma Android (modificado de MARTINS, 2009, p. 5).

A arquitetura da plataforma Android baseia-se no Linux Kernel versão 2.6, como observado na Figura 7 parte A, sendo a camada mais baixa da arquitetura. Esta divisão da arquitetura é responsável por permitir uma abstração entre o *hardware* e *software*, bem como gerenciar os principais serviços do sistema como segurança, memória, processos, *threads*, arquivos, pastas, redes, *drivers* etc.

As aplicações para Android são executadas em uma máquina virtual chamada *Dalvik*, otimizada especialmente para dispositivos móveis, como observado na Figura 7 parte B. Esta máquina virtual apresenta melhor desempenho e maior integração com a nova geração de *hardware*, bem como foi projetada para executar mais de uma máquina virtual de forma paralela. Por ter sido projetada para ser utilizada consumindo o mínimo de memória, bateria e CPU, a *Dalvik* funciona em sistemas com baixa frequência de CPU, pouca memória RAM e sistema operacional sem espaço de *Swap*. A máquina virtual converte o *bytecode* para seu formato especifico, o .dex (*Dalvik Executable*), que representa a aplicação do Android compilada (MARTINS, 2009, p. 5).

Além do sistema operacional e da máquina virtual, a arquitetura da plataforma Android é formada por um conjunto de bibliotecas C/C++, as quais são utilizadas por vários componentes do sistema, como apresentado na Figura 7 parte C. Tais bibliotecas fornecem áudio, vídeo, visualização de camadas 2D e 3D, funções para navegadores *Web*, funções para

gráficos, Banco de Dados etc., sendo disponibilizados por meio do *framework* de aplicação para os desenvolvedores.

A camada *Aplication Framework* (Figura 7 parte D), ou seja, a camada do *framework* de aplicação é composta por programas responsáveis por gerenciar as funções básicas do telefone, envolvendo alocação de recursos, aplicações de telefone, mudança entre processos ou programas. Sendo assim, os desenvolvedores têm a possibilidade de explorar da melhor forma possível a capacidade de *hardware* do dispositivo, executar serviços em *background*, entre outros. A camada em questão oferece ferramentas básicas para a criação de ferramentas mais sofisticadas, visando simplificar a reutilização de componentes. Dentre os principais elementos desta camada, têm-se (PEREIRA & SILVA, 2009, p. 6):

- *activity manager:* é responsável por gerenciar o ciclo de vida de todas as atividades, desde o momento em que uma atividade é iniciada até o seu encerramento;
- *package manager:* é responsável por estabelecer comunicação com o restante do sistema e indicar quais os pacotes que estão sendo utilizados no dispositivo, bem como a capacidade de cada um desses pacotes;
- *windows manager*: é responsável por gerenciar as apresentações de janelas, indicando quais estarão ativas e quais não estarão;
- *content providers*: é responsável por estabelecer a troca de informações entre aplicativos por meio do compartilhamento de dados possibilitado entre os aparelhos;
- *view system:* é responsável por disponibilizar todo o tratamento gráfico para a aplicação, sendo eles botões, *layouts* e *frames*.

Por fim, em seu nível mais alto (Figura 7 parte E), encontram-se as várias aplicações disponibilizadas e elaboradas na linguagem de programação Java, as quais envolvem tanto aplicações originais, desenvolvidas pelo Google, quanto desenvolvidas por terceiros as quais estão instaladas no sistema. As aplicações originais, como cliente de email, calendários, mapas, navegadores, programas de SMS, agendas, gerenciador de contatos e chamadas telefônicas etc. estão disponíveis no topo da pilha da camada, conhecida por *Aplications*. Sendo assim, esta é responsável por possibilitar a iteração do usuário comum com as interfaces dos aplicativos. Já a pilha mais baixa da camada disponibiliza os aplicativos desenvolvidos por terceiros, os quais são acessados apenas por desenvolvedores e fabricantes de *hardware*.

2.3.2. Conceitos sobre desenvolvimento de aplicações Android

Como mencionado anteriormente, as aplicações para Android são implementadas utilizando a linguagem de programação Java. Assim que o código é compilado, este é empacotado em um arquivo com extensão .apk (MARTINS, 2009, p. 6). É por meio deste arquivo que os usuários finais instalam a aplicação em seus dispositivos.

Quando uma aplicação é executada, esta consome um processo próprio, o qual possui uma máquina virtual própria. Além disso, atribui-se um ID de usuário Linux único para cada aplicação, sendo os arquivos de permissões visíveis apenas para a respectiva aplicação.

Uma aplicação é construída por meio da utilização de componentes básicos que podem ser compartilhados. Esse compartilhamento se refere à possibilidade de utilizar os recursos já desenvolvidos de outras aplicações, quando tais recursos possuem permissões para isso. Vale ressaltar que os componentes de uma aplicação são instanciados apenas quando são necessários. Sendo assim, existem quatro tipos principais de componentes Android (MACK, 2010, p. 33):

- atividades ou *activities*: consistem na tela para o usuário, ou seja, é a representação de uma interface visual que comporta usuários escolher uma determinada tarefa e a executar;
- serviços ou *services:* são responsáveis por executar processamentos em segundo plano não possuindo, desta forma, interface visual;
- provedores de conteúdo ou *content providers:* fornecem dados específicos de uma aplicação para outras aplicações;
- receptores de *broadcast* ou *broadcast receivers:* incidem em componentes que ficam ociosos e respondem a eventos.

Destaca-se que os componentes de atividades, serviços e receptores de *broadcast* são acionados por meio de mensagens assíncronas. Já os componentes provedores de conteúdo são acionados quando forem requisitados por um *ContentResolver*. O objeto que contem a mensagem assíncrona dos três primeiros componentes é conhecido como *Intent*, sendo composta pela ação que se deseja executar. Como exemplo tem-se a transmissão de um pedido de uma atividade a qual exibe uma mensagem ao usuário.

Há o tipo de *Intent* explícito e o implícito, apresentados a seguir:

- *intent* explícito: há a definição explícita do componente que deve ser executado;

- *intent* implícito: a definição do componente a ser executado é determinada pelo sistema operacional. Essa seleção é feita por meio de critérios que determina qual componente é mais adequado para responder a intenção do momento.

Todos os componentes da aplicação são declarados em um arquivo de manifesto contido em cada pacote *.apk*. É neste arquivo que as bibliotecas utilizadas são declaradas, bem como as permissões, versão e requisitos. Tais declarações são estruturadas na linguagem XML.

Segundo Martins (2009, p. 7), um processo com um único *thread* é iniciado quando o primeiro componente de uma aplicação precisa ser executado. Ainda segundo o autor, por padrão o mesmo *thread* é responsável por executar todos os componentes da aplicação. Ressalta-se que a chamada de procedimentos remotos, executados em outros processos, é permitida por um mecanismo disposto pelo Android.

Quando um processo é destruído pelo Android, consequentemente todos os componentes da aplicação em execução neste mesmo processo também são destruídos. O que determina a escolha de qual processo deve ser destruído pelo Android é a importância deste para o usuário, excluindo-se, como exemplo, os processos com atividades que não estão mais visíveis ao usuário na tela da aplicação.

As próximas seções apresentarão de uma forma mais aprofundada os dois principais tipos de componentes Android, citados anteriormente, *Activities* e *Services*.

2.3.2.1. Activities

Como apresentado anteriormente, uma atividade consiste na representação da aplicação em forma de interface, possibilitando a iteração com o usuário de forma visual. A criação da interface com o usuário só é possível por meio da composição das atividades, que operam independentes. Isto que dizer que cada atividade é aplicada como subclasse da classe base da atividade, podendo existir diversas atividades dentro de uma aplicação. Cada atividade é representada por uma janela, a qual pode preencher a tela, flutuar por cima de outras janelas ou até mesmo interagir com outras janelas.

Cada componente possui seu ciclo de vida, o qual se inicia a partir do momento que é instanciado e finaliza quando é destruído. Quando em execução, o componente pode ter seu estado alterado várias vezes, representando quando uma atividade está ativa ou não, bem como se está visível ao usuário ou não. Sendo assim, uma atividade possui estados como *ativa*, *pausada e parada*, como apresentado abaixo (MACK, 2010, p. 33):

- *active* ou *running*: quando uma atividade está em primeiro plano pronta para receber as ações dos usuários, pode-se dizer que ela está em estado ativo, estando em primeiro plano nas execuções de tarefas;
- *paused*: quando uma atividade ainda está visível ao usuário, mas sem foco, esta se encontra em estado pausado. Isso acontece quando uma segunda atividade está em execução enquanto uma primeira está parcialmente encoberta ou a segunda atividade é transparente. Mesmo estando em segundo plano, a atividade continua contendo todas as informações obtidas enquanto ativada;
- *stopped*: quando uma atividade está totalmente encoberta por outra atividade, não estando visível ao usuário, esta primeira atividade se encontra em estado pausado. Enquanto não for destruída pelo sistema, a atividade em pausa continua mantendo seus estados e informações.

À medida que uma aplicação muda de estado, é transmitida uma mensagem ao sistema que pode ser contraída pela aplicação. Sendo assim, pode-se estabelecer uma função para cada uma das ações da atividade.

A Figura 8 representa o ciclo de vida de uma atividade.

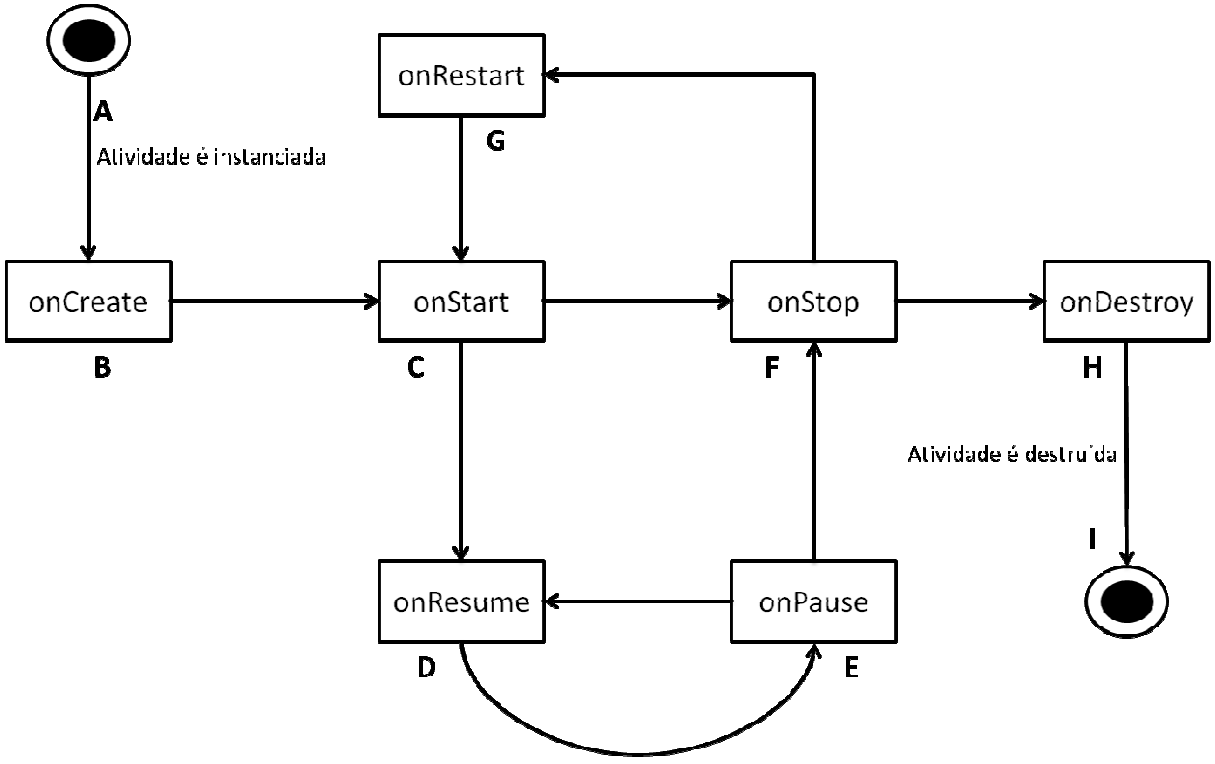

Figura 8: Ciclo de vida de uma atividade (modificado de MARTINS, 2009, p. 8).

A partir do momento que uma atividade é executada (Figura 8 parte A), cria-se uma instância da atividade pela chamada ao método *onCreate* (Figura 8 parte B). Logo após que é criada, a atividade é iniciada pela chamada ao método *onStart* (Figura 8 parte C). Quando iniciada, a atividade pode ser resumida (por meio do método *onResume*, conforme Figura 8 parte D) ou paralisada (por meio do método *onStop*, conforme Figura 8 parte F). Caso a atividade seja resumida, esta é consequentemente pausada pelo método *onPause* (Figura 8 parte E) e, em seguida paralisada (por meio do método *onStop*, conforme Figura 8 parte F) ou resumida novamente, voltando ao método *onResume* (Figura 8 parte D). Agora caso a atividade seja paralisada, esta pode ser reprocessada pela chamada ao método *onRestart* (Figura 8 parte G), a qual retorna ao processo novamente a partir da inicialização da atividade (Figura 8 parte B); ou pode ser destruída pela chamada ao método *onDestroy*, representado na Figura 8 parte H, o qual finaliza a atividade (Figura 8 parte I).

Destaca-se que a atividade pode ser vista pelo usuário durante o tempo entre as chamadas dos métodos *onStart* (Figura 8 parte C) e *onStop* (Figura 8 parte F). O usuário interage com a atividade entre os métodos *onResume* (Figura 8 parte D) e *onPause* (Figura 8 parte E). O método *onPause* (Figura 8 parte E) deve ser usado para registrar dados para futura restauração, de forma durável, uma vez que é o único método chamado obrigatoriamente antes do processo ser eventualmente destruído.

Retomando aos métodos do ciclo de vida representado na Figura 8 e mencionados anteriormente, tem-se: *onCreate(), onRestart(), onStart(), onResume(), onPause(), onStop()* e *onDestroy(),* os quais serão apresentados (AMORIM, 2011, p. 20):

- *onCreate()*: é chamado quando uma atividade é criada pela primeira vez. É neste método que os pontos de vista são criados, dados são vinculados a lista, bem como outras rotinas que fazem parte de todo conjunto estático normal da aplicação. Este método é seguido pelo método *onStart()*;
- *onRestart()*: é chamado assim que a atividade é interrompida, um pouco antes de ser iniciada novamente, sendo seguido pelo método *onStart()*;
- *onStart()*: é chamado logo antes da atividade tornar-se visível para o usuário. Quando a atividade vem para o primeiro plano, o método em questão é seguido pelo método *onResume()*. Já quando se torna oculto, é seguido pelo método *onStop()*;
- *onResume()*: é executado antes que a atividade de interação com o usuário seja iniciada, estando assim no topo da pilha de atividades. Este método é sempre seguido por *onPause()*;

- *onPause()*: é chamado quando outra atividade está quase retomando, estando prestes a retomar ao sistema. Confirmações de alterações não salvas, dados persistentes, animações *stop* e outras ações que consumem CPU são geralmente realizadas neste método. Uma vez que a próxima atividade não é retomada até que o método em questão retorne, este deve concluir todo o processo de forma ágil. No caso da atividade retornar para frente, o método *onPause()* é seguido pelo método *onResume()*. Por outro lado, quando a atividade se torna invisível para o usuário, o método *onPause()* é seguido pelo método *onStop()*;
- *onStop()*: é chamado quando a atividade está sendo destruída ou quando outra atividade foi retomara e está a cobrindo, estando a atividade nestes dois casos invisível ao usuário. No caso da atividade de interação com o usuário esteja voltando, o método em questão é seguido pelo método *onRestart()*. Já quando a atividade está indo embora, o método *onDestroy()* é o próximo a ser executado;
- *onDestroy()*: é chamado antes de uma atividade ser destruída, sendo a última chamada que uma atividade recebe. Portanto, este método não é seguido por nenhum outro método.

Além dos *activities*, têm-se os *services* como outro tipo de componente Android, o qual será apresentado na próxima seção.

2.3.2.2. Services

Como apresentado anteriormente, os serviços são executados em segundo plano por tempo indeterminado, não possuindo interface visual para o usuário. Sendo assim, o serviço se mantém ativo até que receba outra ordem. Como exemplo tem-se o *player* de mídia, o qual permite que o usuário escolha músicas e as toque, podendo executar outros serviços enquanto a música escolhida é executada. Quando o usuário inicia uma música, o serviço é responsável por fazer com que ela continue tocando até que o usuário indique outra atividade como, por exemplo, parar a música ou passar para uma próxima. A partir do exemplo dado, pode-se dizer que um serviço continua a rodar em segundo plano quando outro serviço é iniciado, mesmo quando o usuário passa a utilizar outro aplicativo (AMORIM, 2011, p. 66). Segundo Mack (2010, p. 34), os serviços são executados no *thread* principal do processo da aplicação.

De forma similar às atividades, “os serviços possuem estados e métodos de ciclo de vida que podem ser implementados para responder a mudanças de estado” (MARTINS, 2009, p. 8). Os estados dos serviços podem ser essencialmente de duas formas: iniciado e ligado (AMORIM, 2011, p. 66). Esses dois estados são descritos a seguir:

- iniciado: o serviço é iniciado através da chamada ao método *startService()*, o qual é invocado quando um componente da aplicação inicia o serviço. O serviço assim que é iniciado, pode ser executado em segundo plano por tempo indefinido até mesmo se o componente que o iniciou é destruído. Neste estado não ocorre retorno de resultado para o chamador. Sendo assim, o serviço iniciado é executado em uma única operação;
- ligado: o serviço é ligado através da chamada ao método *bindService()*, o qual é invocado quando um componente da aplicação liga-se a ele. É neste estado que é estabelecida uma interface cliente-servidor, possibilitando desta forma o envio de pedidos, obtenção de resultados, dentre outras ações. Vale ressaltar que um serviço pode ter ligado a si vários componentes de aplicação de forma simultânea. Sendo assim, quando o serviço não tiver nenhum componente ligado a ele, este é destruído.

Vale ressaltar que um serviço pode possuir seus dois possíveis estados em um único momento, ou seja, pode ser iniciada, ligada ou ambos. A situação de um serviço funcionar nos dois sentidos ocorre devido ao fato deste poder ser iniciado para executar indefinidamente como também permitir a ligação.

Destaca-se ainda que qualquer componente de aplicativo pode usar um serviço, independente da aplicação estar iniciada, ligada ou ambos (AMORIM, 2011, p. 67). Igualmente, qualquer componente pode utilizar uma atividade por meio de uma *Intent*. Todavia, um serviço pode ser bloqueado para que outros aplicativos não tenham acesso ao mesmo por meio de sua declaração como privado.

Segundo Amorim (2011, p. 68), um serviço deve ser criado a partir de uma subclasse de *Service*. Ainda segundo o autor, quando em execução, há quatro métodos de retorno mais importantes, os quais são listados a seguir:

- onStartCommand(): é chamado quando um componente requer que o serviço seja iniciado. Quando iniciado, este serviço pode rodar em segundo plano por tempo indefinido. Mas vale ressaltar que fica da responsabilidade do desenvolvedor do aplicativo parar o serviço, que pode ser feito através da chamada aos métodos stopSelf() ou stopService();
- onBind(): é chamado quando um outro componente quer se vincular ao serviço. Este vínculo é estabelecido através da chamada ao método bindService(). Este método necessita de uma interface com o cliente, a qual é utilizada para retornar um resultado, fornecendo desta forma comunicação entre o serviço e o cliente;

- onCreate(): é chamado assim que o serviço é criado. Este método é responsável por executar os procedimentos de configuração. O único caso em que este serviço não é chamado ocorre quando o serviço já está em execução;
- onDestroy(): é chamado quando o serviço está sendo destruído por não estar sendo mais utilizado, sendo assim a última chamada ao serviço. Este método é importante para a limpeza de recursos como *threads*, receptores, entre outros.

Amorim (2011, p. 69) afirma que quando o sistema Android está com pouca memória, este deve recuperar os recursos do sistema para a atividade que tem o foco do usuário, sendo tal recuperação realizada forçando a parada de um serviço. O autor destaca ainda que um serviço dificilmente é finalizado quando está vinculado a uma atividade que tem o foco do usuário, bem como quando é declarado a ser executado em primeiro plano.

2.3.3. Interface com o usuário

A interface de um sistema define os aspectos de comunicação com o usuário. Segundo Amorim (2011, p. 108), as unidades principais de demonstração da interface do usuário na plataforma Android são os objetos *View*, que envolvem campos de texto e botões. Já as diferentes estruturas de *layout* de arquitetura, tais como a linear, tabular e relativa são implementadas pelos objetos *ViewGroup*.

Como mencionado, a interface com o usuário no Android é construída por meio de uma hierarquia de objetos, representados em forma de árvore, em que as folhas são do tipo *View* e os ramos do tipo *ViewGroup* (MARTINS, 2009, p. 8). A Figura 9 representa a árvore de objetos da interface do Android.

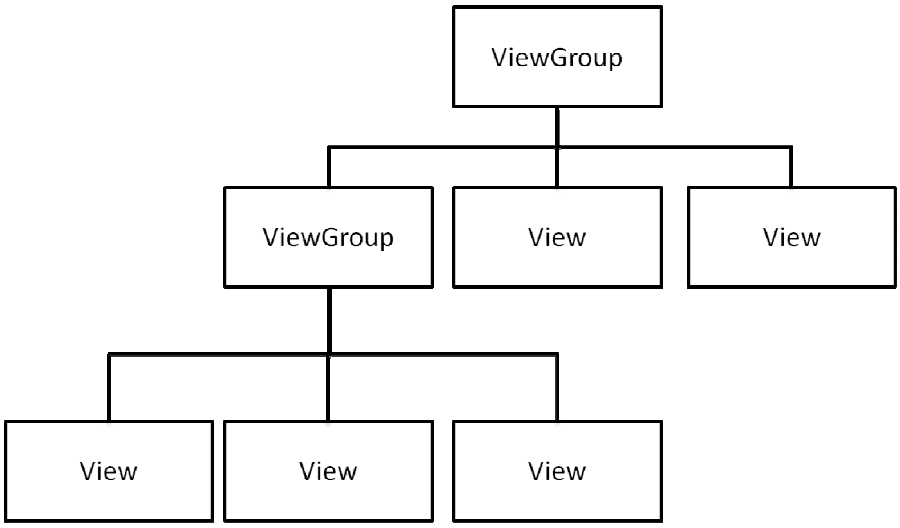

Figura 9: Árvore de objetos *View* e *ViewGroup* (modificado em MARTINS, 2009, p. 9).

Como pode ser observado na Figura 9, os ramos da árvore de interface do Android são chamadas de *ViewGroup* e as folhas *View*. "Os objetos podem ser organizados automaticamente, dentro de uma região especifica da tela, entre outras, de forma linear, relativa ou absoluta" (MARTINS, 2009, p. 9). A atividade de uma tela deve chamar o método *setContentView()* e passar uma referência ao objeto nó raiz para fixar a árvore de hierarquia de *Views* à tela para renderização (AMORIM, 2011, p. 109). Ao receber a referência, o sistema Android invalida, mede e desenha a árvore.

Ainda segundo o autor, apesar da possibilidade de definir as coordenadas e o tamanho do componente a ser desenhado na tela, esta funcionalidade deve ser evitada. Ao contrário, julga-se proveitosa a utilização da disposição em *layouts* não absolutos, devido ao fato de permitir um melhor ajustamento da interface da aplicação às distintas telas dos dispositivos no quais pode ser executada, sendo essas diferentes telas referentes à resolução e tamanho distintos.

Através da plataforma Android é possível criar ricas interfaces com o usuário com a utilização de *widgets*, componentes gráficos que flutuam pela tela e fornece uma funcionalidade específica para o usuário, e *layouts* prontos, além de ser possível estabelecer e acrescentar elementos como caixas de textos, botões, dentre outros, aplicando-se estilos e temas.

Recursos de animação gráfica em 2D também são disponibilizados pela plataforma Android, tornando o visual da aplicação ainda mais atrativo. Como exemplo de animação tem-se a chamada quadro a quadro, que consiste na troca de imagens especificas de forma

ordenada conforme especificado pelo desenvolvedor. Outro exemplo de animação é a chamada *Tween*, que consiste na transformação simples em um objeto *View*, que envolvem tamanho, posição, transparência e rotação.

Além dos recursos apresentados anteriormente, a plataforma Android disponibiliza ainda a criação de menus personalizados, existindo três fundamentais tipos de menus: opção, submenu e contexto, apresentados abaixo:

- opção: é exibido quando a tecla MENU do dispositivo é pressionada;
- submenu: é exibido quando um de seus itens de nível acima é selecionado;
- contexto: é exibido quando um objeto *View* é pressionado longamente.

Existem três técnicas disponíveis no Android que podem ser utilizadas para notificar o usuário sobre um evento que tenha ocorrido na aplicação, sendo eles: *Toast*, diálogo ou barra de estado, apresentados a seguir:

- *toast:* é utilizada no caso da não necessidade de interação como, por exemplo, para mensagens pequenas;
- diálogo: é utilizado para capturar uma resposta por atrair a atenção do usuário;
- barra de estado: é utilizado no caso da aplicação necessitar notificar o usuário quando está sendo usada em segundo plano. Neste caso, uma vez que a aplicação tenha permissão para emitir sons ou vibrar o dispositivo, tais recursos podem ser utilizados neste tipo de técnica em questão.

Tendo entendido todos os conceitos de desenvolvimento para aplicações Android e a forma pela qual são elaboradas as interfaces destes aplicativos, a próxima seção apresentará como pode ser realizado o armazenamento de informações que aplicativos Android podem disponibilizar.

2.3.4. Armazenamento de Informações

O armazenamento de informações em aplicações Android pode ser realizado por meio de um mecanismo conhecido como preferências. Este mecanismo permite o armazenamento de tipos primitivos, sendo normalmente utilizado para armazenar as preferências do usuário.

Outra forma de armazenamento de informações em aplicações Android é por meio de arquivos, utilizando desta forma o armazenamento no próprio aparelho ou em memória removível. Tem-se ainda a possibilidade de armazenar informações em tabelas e em Banco de Dados, com o SQLite, que consiste em uma biblioteca na linguagem C que implementa um Banco de Dados SQL embutido. Além disso, tem-se a possibilidade de armazenar e/ou acessar informações pelo acesso a operações de rede.

O Banco de Dados SQLite apresenta características como ser público e de código aberto, sendo um banco relacional e de simples utilização. Este Banco de Dados não necessita de configuração, utilizando desta forma as configurações do próprio sistema para o controle de acesso a dados. O SQLite armazena em um único arquivo a estrutura de dados completa como tabelas, *views*, índices e *triggers*.

3 MATERIAIS E MÉTODOS

Esta seção apresenta materiais, metodologia tecnologias utilizados no desenvolvimento deste trabalho.

3.1. Materiais

O material utilizado neste projeto foi obtido por meio de pesquisa bibliográfica através da Internet, além de informações oriundas da base de dados do Detran-TO. Entre os materiais obtidos na pesquisa estão inclusos: monografias de graduação, teses de doutorado, dissertações de mestrado, livros, publicações científicas, artigos dentre outros.

3.2. Metodologia

O desenvolvimento deste trabalho iniciou a partir de uma pesquisa em diversos sites para encontrar materiais tais como livros, artigos, monografias, teses, dissertações, dentre outros. Tais materiais serviram como referência bibliográfica para os principais assuntos envolvidos que resultaram na compreensão e aplicabilidade dos conceitos adquiridos no desenvolvimento do trabalho, que compreendem *Web Services*, SOAP e Android.

Uma vez finalizado este levantamento bibliográfico, foi realizado um estudo mais aprofundado sobre a plataforma Android, a fim de verificar as tecnologias existentes para a criação de uma aplicação que recupere dados de veículos e condutores no Detran-TO para auxiliar a Polícia Militar nas blitz executadas, com auxílio de *Web Services*. Dentre as ferramentas encontradas tem-se o ADT Plugin, KSoap e Android SDK, sendo este último o *kit* de desenvolvimento para aplicações baseadas no Sistema Operacional Android. Adotou-se ainda a IDE (*Integrated Development Environment*) Eclipse, para o desenvolvimento do aplicativo, devido às seguintes características:

- possuir suporte para executar aplicativos baseados em Android SDK;
- possuir um *plugin* customizado conhecido por Android *Development Tool* (ADT);
- ser uma ferramenta popularmente utilizada por desenvolvedores de forma geral.

Tanto as ferramentas quanto as tecnologias utilizadas para o desenvolvimento do presente trabalho, serão apresentadas a posteriori, na Seção 3.2.1.

A partir do momento em que os conceitos teóricos foram adquiridos e escolhidos os recursos de implementação necessários no desenvolvimento do trabalho, iniciou-se a criação da aplicação. Foi desenvolvido então um *Web Service* para o acesso à base de dados do Detran-TO, a fim de obter informações referentes a veículos e condutores do Brasil e do Estado do Tocantins, respectivamente.

Por fim, finalizado o processo de comunicação do *Web Service* com a base de dados do Detran-TO, iniciou-se a implementação do aplicativo. A execução dessa atividade possibilitou a consulta de informações por meio de um dispositivo com a plataforma Android. Tais informações abrangem dados referentes aos veículos de todo o País, contendo desde mandado de busca e apreensão e furto/roubo, até informações básicas como chassi, proprietário, pagamento de licenciamento do ano corrente, entre outros. Além de informações dos veículos, o aplicativo disponibiliza informações relacionadas aos condutores do Tocantins, permitindo ao usuário do sistema acesso às informações completas do condutor através da base de dados do Detran-TO.

As próximas seções apresentarão os recursos utilizados no desenvolvimento deste trabalho, que envolvem ferramentas, APIs e *plugins*, citados anteriormente.

3.2.1. Ferramentas e Tecnologias

O Android SDK consiste em um kit de desenvolvimento que oferece ferramentas e APIs que auxiliam na implementação de aplicativos para a plataforma Android, através da utilização da linguagem de programação Java. As ferramentas disponibilizadas pelo Android SDK envolvem emulador, *plugin* para o Eclipse, entre outras.

O emulador é responsável por representar o funcionamento de um telefone celular baseado na plataforma Android, executando assim alguns recursos que podem ser executados em um dispositivo físico com Android, mas sem a necessidade deste equipamento. É por meio do emulador que testes nos códigos são realizados, bem como instaladas e/ou executadas aplicações destinadas à plataforma Android. Desta forma geral, o emulador garante a consistência do *software*, uma vez que o aplicativo terá o mesmo funcionamento no aparelho, pois simula tanto *hardware* quanto *software*.

O Android SDK pode rodar nas IDEs Eclipse, Intellije e Netbeans (PEREIRA & SILVA, 2009, p. 17). Sendo assim, o Android SDK disponibiliza um *plugin* para a IDE

Eclipse, que é uma ferramenta para o desenvolvimento de sistemas na linguagem de programação Java, a qual consiste em uma linguagem sólida e bem aceita no mercado. Este *plugin* facilita a codificação do aplicativo, por fornecer soluções e interface gráfica, acrescenta apoio integrado com o projeto e ferramentas Android, agilizando e facilitando o desenvolvimento por proporcionar a construção de extensões, a execução de depuração e do aplicativo (PROJECT, 2009, online, tradução nossa).

Para realizar a integração entre o Eclipse com o emulador, tem-se o *plugin Android Development Tools* (ADT), a qual incorpora as ferramentas de desenvolvimento ao Eclipse. Este *plugin* dispõe de um conjunto de ferramentas que visa facilitar o desenvolvimento, possibilitando a criação ágil de um sistema *Android* com interface ao usuário e com componentes baseados no *Framework* do *Android* (LIMA et. al., 2011, p. 40).

O Apêndice A apresenta todas as informações necessárias para a configuração das ferramentas utilizadas no desenvolvimento do aplicativo Android apresentado neste trabalho.

4. RESULTADOS E DISCUSSÕES

Este trabalho tem como objetivo desenvolver um aplicativo que auxilie a Polícia Militar do Estado do Tocantins em suas fiscalizações conhecidas como “blitz”. Estas fiscalizações têm o objetivo de aferir a documentação do condutor e veículo, constatando que ambos estão com todas as suas obrigações cumpridas juntamente à União. Este auxílio está relacionado à disponibilidade de informações referentes aos condutores e veículos do Estado do Tocantins e Brasil em tempo real, que abrangem legitimidade dos documentos dos condutores, pendências junto ao Detran, roubo/furto, mandado de apreensão, características do veículo, dentre outras.

A proposta de desenvolver este trabalho surgiu da necessidade de se ter mais informações acerca de veículos e condutores por parte da Polícia Militar, e que estas informações fossem acessadas de forma rápida e prática. A partir desta premissa, foram utilizados recursos já existentes no Detran-TO para a solução deste problema. Dentre estes recursos encontram-se *Tablets* com Sistema Operacional Android e a presença de *Web Services* já desenvolvidos no Órgão. A partir destes recursos, a aplicação desenvolvida utiliza os *Web Services* já existentes e apresenta os resultados ao usuário em *Tablets* e smartphones, já utilizados anteriormente por funcionários do Detran-TO para outros fins.

As informações apresentadas aos usuários da aplicação Android desenvolvida foram escolhidas com base nos Web Services já criados. Vale destacar que estes Web Services são utilizados em outras aplicações no Detran-TO, atendendo aos usuários com todas as informações necessárias acerca de veículos e condutores, logo, estas informações são suficientes para a Polícia Militar executar a aferição das informações em blitz de forma satisfatória.

De forma geral, as informações que o aplicativo disponibiliza será abstraídas das Bases de Dados dos sistemas do Detran do Estado do Tocantins. Nas consultas que necessitam de recuperação de dados oriundas de sistemas externos ao Detran-TO, os serviços construídos pelo Detran-TO possuem autonomia para comunicação com estes sistemas, possibilitando a recuperação de dados necessários de outros centros de dados como Base Nacional, Sefaz-TO, Sistema de Gravame e sistemas de outros estados. Com o auxílio desta aplicação, as fraudes de documentos serão mais fáceis de serem detectadas, visto que o usuário poderá confrontar os dados existentes nos documentos do condutor e do veículo com

os dados retornados pelo Detran-TO através da aplicação, podendo também verificar dados que não existem nos documentos impressos de porte do condutor, como ocorrência de roubo/furto e ordem de apreensão de veículo.

O aplicativo está baseado na plataforma Android e pode ser instalado em *Smartphones*, *Tablets* ou qualquer outro componente que possuir Sistema Operacional Android. A aplicação se comunica com o Banco de Dados do Detran-TO, o mesmo utilizado nas aplicações do Dentran-TO pelos seus funcionários. Como a extração de informações é realizada através da Base de Dados do Detran-TO, a qual é utilizada pelos sistemas que o órgão possui, as informações processadas nos dois ambientes (sistemas do Detran-TO e aplicativo Android) são as mesmas. Sendo assim, no momento que houver uma inserção de dados no sistema do Detran-TO como, por exemplo, a aplicação de uma multa a um determinado veículo, esta poderá ser verificada imediatamente no aplicativo Android.

Caso um veículo possua uma ordem de apreensão em qualquer Estado, ao ser pesquisado no aplicativo Android será possível visualizar a informação de que o veículo apresenta uma ordem de apreensão, possibilitando ao policial detentor desta informação tomar as medidas cabíveis.

A comunicação entre o aplicativo Android e a Base de Dados do Detran-TO é realizada através de *Web Services* já disponibilizados por este órgão, sendo estes *Web Services* responsáveis por executarem procedimentos no Banco de Dados do Detran-TO, recuperando informações que são enviadas para apresentação na aplicação Android, apresentando ao usuário todos os dados solicitados. Havendo a necessidade de busca de informações adicionais em outras Bases de Dados, os procedimentos executados pelos *Web Services* farão a comunicação com as Bases de Dados responsáveis por deterem as informações adicionais necessárias, sendo estas a Base Nacional, Sefaz-TO, Sistema Nacional de Gravas e os outros estados da união.

A Figura 10 apresenta a estrutura da comunicação do aplicativo proposto.

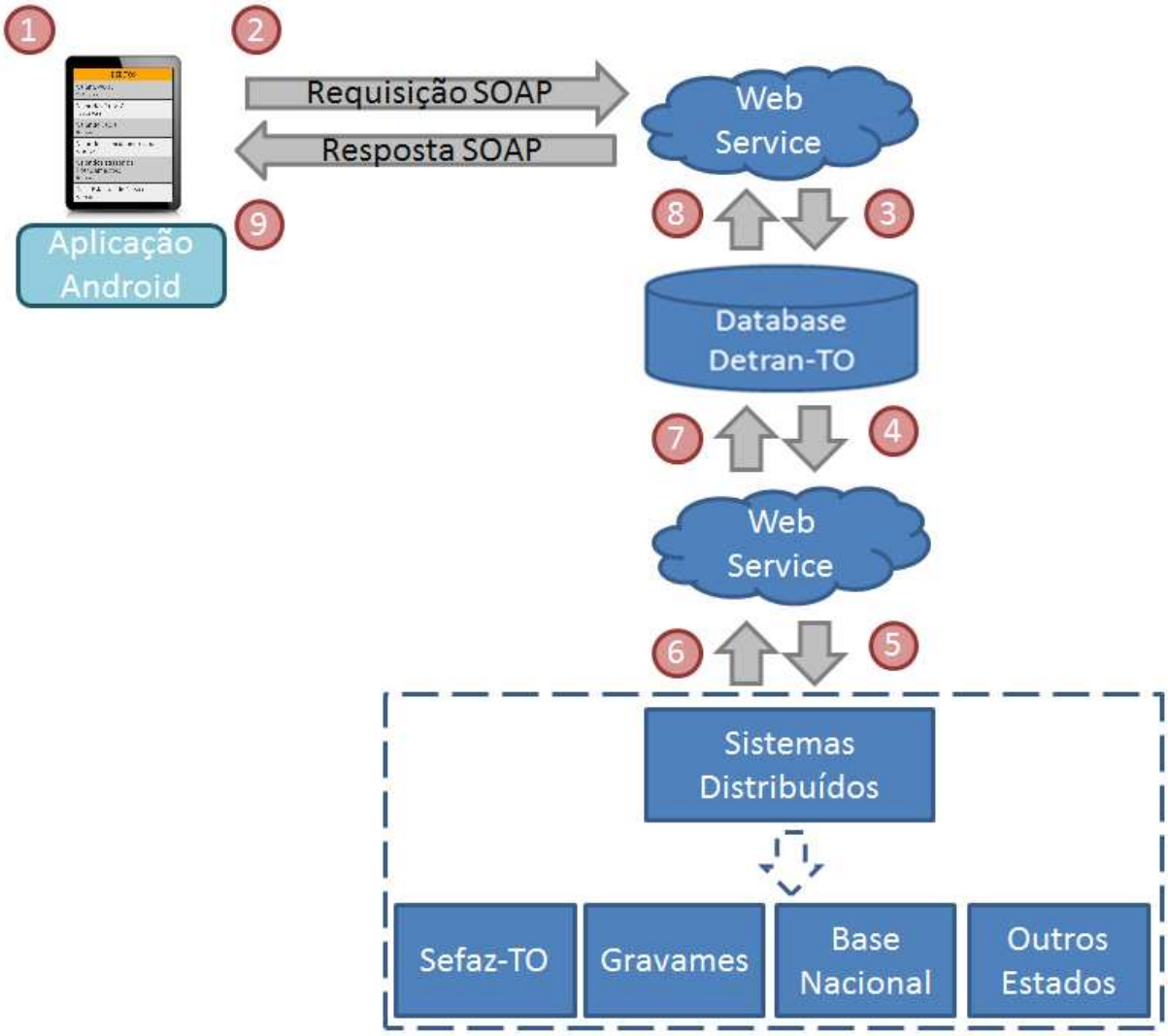

Figura 10: Fluxo da comunicação entre a aplicação Android e Detran-TO.

O fluxo da Figura 10 tem seu início no momento em que o usuário da aplicação Android entra com os dados solicitados no dispositivo, representado pelo passo 1. Finalizada a inserção de dados, a aplicação envia as informações por meio de *Web Service* mediante auxílio do protocolo SOAP para o Detran-TO, visto no passo 2. O passo 3 demonstra a ação do *Web Service*, responsável por fazer a chamada da *procedure* no Banco de Dados do Detran-TO referente ao serviço solicitado pelo usuário através da aplicação Android. Caso seja necessário, a *procedure* solicitará informações aos sistemas distribuídos ligados ao Detran-TO através de um *Web Service*, como demonstrado nos passos 4 e 5. Após obter a informação dos sistemas distribuídos, este retornará os dados, demonstrado nos passos 6 e 7, sendo tratados na *procedure* do Banco de Dados do Detran-TO, retornando o resultado à aplicação Android, visto nos passos 8 e 9.

O fluxo de dados apresentado na Figura 10 é a base para todas as comunicações entre a aplicação Android e a base de dados do Detran-TO, sendo estas listadas a seguir:

- autenticação;
- consultar débitos de veículo;
- consultar informações de veículo no Tocantins;
- consultar informações de veículo na Base Nacional;
- consultar informações de condutor.

Tendo o modelo apresentado na Figura 10 como base, as próximas seções apresentam os fluxos de dados referentes a cada processo constituído na aplicação Android de forma detalhada.

4.1. Autenticação

Para ter acesso a todos os procedimentos e informações existentes na aplicação Android, o usuário deve se autenticar informando seus dados, sendo verificada sua veracidade.

Atualmente, o Detran-TO possui um *Active Directory* (AD), que tem registro de todos os funcionários do Órgão. Este AD é responsável por autenticar todos os usuários em todos os sistemas existentes no Órgão, ou seja, só tem acesso aos sistemas do Detran-TO os usuários que tiverem credenciais neste AD. Sendo assim, a autenticação da aplicação Android é feita junto ao AD do Detran-TO, obrigando o usuário da aplicação a possuir uma conta ativa. Os dados enviados pelo *Web Service* relacionados à autenticação do usuário são apresentados na Tabela 2.

Tabela 2 – Dados enviados pelo *Web Service* de Autenticação.

Item	Elemento	Propriedade
1	CPF	Int
2	Senha	String

O item 1 da Tabela 2 representa o CPF do usuário, sendo este um inteiro de 11 dígitos. O item 2 representa a senha do usuário, contendo 20 caracteres em seu total. A Tabela 3 apresenta os dados retornados pelo *Web Service*, informando a validade das informações enviadas de acordo com a Tabela 2.

Tabela 3 – Dados recebidos pelo Web Service de Autenticação.

Item	Elemento	Propriedade
1	Confirmação	String
2	Descricao	String

O item 1 da Tabela 3 representa a confirmação da autenticação feita pelo *Active Directory*. O universo deste item é resumido em *"S"* quando a autenticação for realizada com sucesso ou *"N"* para a autenticação não ter sido realizada com sucesso. Já o item 2 representa a descrição do item 1, informando a razão do usuário ter ou não conseguido a autenticação. Normalmente, o insucesso da autenticação se deve a erros nos dados informados, no caso do usuário não possuir acesso ou devido ao usuário estar com suas credenciais expiradas. Caso não consiga efetuar a autenticação, é exibido na tela do aplicativo o motivo para que o usuário tome providências para ter seu acesso ao sistema regularizado.

A comunicação estabelecida entre a aplicação Android e o Detran-TO através do *Web Service* pode sofrer algum tipo de problema, sendo disparadas exceções que informarão ao usuário os problemas encontrados no estabelecimento da comunicação entre os pontos origem e destino. As exceções que podem ser disparadas caso algum problema aconteça na comunicação realizada nesta autenticação são várias, sendo que as mais comuns são: o *TimeOut* na comunicação com os servidores do Detran-TO, ocasionado pela queda dos servidores do Detran-TO; *gateway* desconexo; limitação da banda; Banco de Dados fora de operação; demora da resposta do *Web Service*; entre outros. Este erro é apresentado para o usuário, sendo informado qual o tipo de exceção gerada. Não havendo conectividade com a Internet, é exibida uma mensagem informando o erro e solicitando a verificação do problema por parte do usuário.

4.1.1. Desenvolvimento da Autenticação

Os códigos relatados nesta seção exemplificam como foi implementada a autenticação do usuário a fim de proporcionar-lhe credenciais para acesso às funcionalidades do sistema, demonstrado todo o processo de autenticação que as informações do usuário serão submetidas. O processo tem seu início representado pela Figura 11 e a Listagem 2.

Figura 11: Layout de Autenticação.

A Figura 11 apresenta a tela de autenticação ao usuário. É através desta interface que o usuário apresenta seus dados e efetua a autenticação na aplicação. A Listagem 2 apresenta o layout da Figura 11 em forma de código, ou seja, a interface por ser feita de duas formas, sendo criada a partir de um código fonte, como na Listagem 2, ou criando o layout a partir de componentes gráficos, como na Figura 11.

A Figura 11 é a representação gráfica do que foi construído em XML pela Listagem 2, ou seja, a *interface* construída através do código da Listagem 2 é representada graficamente pela Figura 11.

```
<ScrollView android:layout_height="wrap_content" android:id="@+id/scrollView1">
    <TableLayout android:layout_height="wrap_content" android:layout_width="fill_parent">

        <ImageView
            android:id="@+id/imageView1"
            android:layout_height="wrap_content"
            android:background="@drawable/icon_veiculo"
            android:src="@drawable/icon_autenticador" android:adjustViewBounds="true"/>
        <TextView
            android:gravity="center"
            android:id="@+id/textView2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Usuário (CPF)"
            android:textAppearance="?android:attr/textAppearanceLarge" />
        <EditText android:gravity="center"
            android:layout_weight="1"
            android:id="@+id/cpf"
            android:maxLength="11"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:capitalize="characters"
            android:hint="CPF"
            android:numeric="integer"/>
        <TextView
            android:gravity="center"
            android:id="@+id/textView2"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="Senha"
            android:textAppearance="?android:attr/textAppearanceLarge" />
        <EditText android:gravity="center"
            android:layout_weight="1"
            android:id="@+id/senha"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:capitalize="none"
            android:hint="SENHA"
            android:inputType="textPassword"/>
        <TextView
            android:gravity="center"
            android:id="@+id/erro"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text=""
            android:textColor="@color/laranja"/>
        <Button android:keepScreenOn="true"
            android:layout_width="fill_parent"
            android:text="Entrar"
            android:id="@+id/entrar"
            android:gravity="center"
            android:layout_height="wrap_content"
            android:layout_weight="1"/>
        <Button
            android:keepScreenOn="true"
            android:layout_width="fill_parent"
            android:text="Sair do Sistema"
            android:id="@+id/sair"
            android:gravity="center"
            android:layout_height="wrap_content"
            android:layout_weight="1"/>
    </TableLayout>
</ScrollView>
```

Listagem 2: Layout de Autenticação.

Conforme Listagem 2, as linhas 4 a 8 representam a imagem que é apresentada ao usuário referente ao Detran-TO. As linhas 9 a 15 representam o texto “Usuário (CPF)” que é apresentado referente ao campo *EditText* abaixo. As linhas 16 a 24 se referem à construção do campo texto responsável pela entrada de dados do CPF do usuário.

As linhas 25 a 31 apresentam o texto “Senha”, informando ao usuário que o campo abaixo é responsável por conter a senha para autenticação. As linhas 32 a 39 apresentam o campo de entrada de dados referente à senha do usuário, armazenando esta informação.

Caso haja algum problema na comunicação com o Detran-TO, este é relatado no campo de texto que foi construído nas linhas 40 a 46, na cor laranja, destacando-se para melhor visualização do usuário. As linhas 47 a 53 representam o botão que faz a validação das informações junto ao Detran-TO. Caso o usuário deseje sair da aplicação, o botão apresentado pelas linhas 54 a 61 possibilita esta ação.

A Listagem 3 é responsável por conter os códigos para que a validação no *layout* da Listagem 2 e Figura 10 aconteça.

```
1  public class Autenticador extends Activity{
2
3      private Button Entrar, Sair;
4      private EditText CPF, senha;
5      private TextView erro;
6      Context context;
7
8      ProgressDialog dialog;
9
10     @Override
11     public void onCreate(Bundle savedInstanceState){
12         super.onCreate(savedInstanceState);
13         setContentView(R.layout.autenticador);
14
15         context = getApplicationContext();
16
17         CPF = (EditText)findViewById(R.id.cpf);
18         senha = (EditText)findViewById(R.id.senha);
19         Entrar = (Button)findViewById(R.id.entrar);
20         Sair = (Button)findViewById(R.id.sair);
21         erro = (TextView)findViewById(R.id.erro);
22
23         ...
24     }
25 }
```

Listagem 3: Início do processo de autenticação.

O processo de autenticação inicia na linha 1 da Listagem 3, com a declaração da classe `Autenticador` sendo herdeira da classe `Activity`, proporcionando que esta classe seja vinculada a uma tela que pode ser mostrada ao usuário, tendo entrada e saída de dados.

As linhas 3 a 6 representam a declaração de variáveis que são utilizadas na classe. A linha 11 apresenta o método que faz a criação da aplicação. Para esta classe ser apresentada como uma tela passível de manipulação pelo usuário, esta deve ser vinculada a um *layout* previamente desenvolvido. Este vínculo é estabelecido conforme a linha 13, sendo `autenticador` o *layout* responsável pela interação com o usuário, sendo localizado na pasta `layout` no projeto criado.

O contexto da aplicação é inserido na variável `contexto`, representando a tela atual em que a aplicação se encontra. As linhas 17 a 21 são responsáveis por variáveis da classe `Autenticador` serem vinculadas aos objetos presentes no *layout* que foi vinculado a esta classe, conforme linha 13.

A Listagem 4 apresenta o código referente ao evento *click* do botão, decorrente da Listagem 3.

```
1  Entrar.setOnClickListener(new View.OnClickListener(){
2      public void onClick(View v) {
3
4          erro.setText("");
5
6          if(internetConnection(context)){
7
8              String SOAP_ACTION = "AÇÃO";
9              String METHOD_NAME = "AUTENTICACAO";
10             String NAMESPACE =   "NAMESPACE_WEB_SERVICE";
11             String URL =         "URL_WEB_SERVICE";
12
13             SoapObject Request = new SoapObject(NAMESPACE, METHOD_NAME);
14
15             Request.addProperty("User", CPF.getText().toString());
16             Request.addProperty("Password", senha.getText().toString());
17
18             SoapSerializationEnvelope soapEnvelope =
19                 new SoapSerializationEnvelope(SoapEnvelope.VER11);
20             soapEnvelope.setOutputSoapObject(Request);
21
22             AndroidHttpTransport aht = new AndroidHttpTransport(URL);
23
24             ...
25         }else{
26             erro.setText("Não foi possível estabelecer conexão com os servidores"+
27             "do DETRAN-TO!\nFavor verificar sua conexão com a internet!");
28         }
29     }
30 }
```

Listagem 4: Parâmetros para conexão com Web Services.

O botão para validação do formulário de autenticação do usuário é definido como `Entrar`, como demonstrado na linha 49 da Listagem 2. A linha 1 da Listagem 4 define a ação que este botão executa após ser pressionado.

A linha 6 verifica se há conexão com a Internet no contexto atual da aplicação. Este método será apresentado na seção 4.6. Havendo retorno positivo de conexão com a Internet, as linhas 8 a 11 se responsabilizam pelos dados referente à localização do serviço de autenticação disponibilizado, sendo necessário a ação, o método de autenticação, *namespace* e a URL do *Web Service*.

Sendo conhecidos os dados sobre o *Web Service* a ser consultado, a linha 13 é responsável por criar um objeto do tipo `SoapObject` com os parâmetros anteriormente solicitados, sendo passados o *namespace* e o nome do método a ser chamado.

O serviço desenvolvido exige dois parâmetros de entrada, sendo eles `User` e `Password`. Estes parâmetros devem ser adicionados no objeto `Request` através do método `addProperty()`, sendo passados os campos estabelecidos no *layout* da aplicação, como visto nas linhas 15 e 16. As linhas 18 a 20 são responsáveis pela criação do objeto que receberá a resposta com base no objeto `Request`. Esta resposta contém os dados de autenticação do usuário.

A linha 22 cria o objeto `aht`, que é responsável pelo transporte HTTP da informação, tendo como parâmetro o endereço do *Web Service*. Não conseguindo acesso à Internet, as linhas 25 a 28 apresentam o erro referente ao não estabelecimento de conexão com os servidores do Detran-TO. A Listagem 5 apresenta o código que fará a chamada do *Web Service*.

```
1  try{
2      aht.call(SOAP_ACTION, soapEnvelope);
3
4      String result = (soapEnvelope.getResult()).toString();
5
6      if(getConfirmacao(result)){
7          erro.setText("Aguarde...");
8
9          Intent myIntent = new Intent(v.getContext(), Principal.class);
10         startActivityForResult(myIntent, 0);
11
12     }else{
13         erro.setText("CPF e/ou Senha informados estão incorretos!"+
14             "\nInforme os dados corretamente para acesso ao sistema!");
15     }
16
17 }catch(SocketTimeoutException e){
18     erro.setText("Servidor está fora de serviço!"+
19         "\n Tente novamente em alguns minutos!");
20 }catch(Exception e){
21     erro.setText(e.getMessage());
22 }
```

Listagem 5: Estabelecendo conexão com Web Services.

Feita toda a preparação para o estabelecimento da conexão com o *Web Service*, a linha 1 da Listagem 5 prepara o código a seguir para eventuais exceções. A linha 6, efetivamente, executa a chamada ao serviço através do método `call()` do objeto `aht`, passando como parâmetro a ação a ser executada e o objeto `soapEnvelope`, responsável por conter o resultado da execução. Na linha 4, a variável `result` recebe, em forma de texto, o resultado retornado pelo servidor do Detran-TO.

Sendo a autenticação bem sucedida com o Detran-TO, como visto na linha 6, é possível ter acesso ao sistema. Destaca-se que a descrição do método da linha 6 é explicado na Listagem 7. A linha 9 é responsável por criar um objeto do tipo `Intent` que referencia a classe `Principal.class`, sendo esta a responsável por apresentar a tela principal do sistema ao usuário, contendo o menu principal da aplicação. O fato de ser necessária a utilização `.class` é uma necessidade da linguagem Java, como forma de indicar um tipo.

Para que esta tela seja apresentada, o método `startActivityForResult()` é executado passando o objeto criado anteriormente, conforme linha 10.

Caso o servidor AD do Detran-TO retorne um insucesso por parte da autenticação, a mensagem de insucesso é apresentada ao usuário conforme linhas 12 a 15. Ainda, havendo uma espera excessiva por parte do objeto `aht`, localizado na linha 2, uma exceção de

TimeOut é lançada, informando ao usuário que este erro ocorreu, conforme linhas 17 a 19. Caso haja outro erro qualquer, este é apresentado conforme linhas 20 a 22. O método `getConfirmacao()` é apresentado na Listagem 6.

```
1  public boolean getConfirmacao(String result){
2      boolean confirmacao = false;
3      String confirmacaoTexto = "";
4      int inicio, fim;
5
6      inicio = result.indexOf("Confirmacao");
7      fim = result.indexOf(";", inicio);
8      inicio = fim - 1;
9      confirmacaoTexto = result.substring(inicio,fim);
10
11     if (confirmacaoTexto.equals("S")){
12         confirmacao = true;
13     }
14
15     return confirmacao;
16 }
```

Listagem 6: Método getConfirmacao().

O método apresentado na Listagem 6 faz a checagem dos dados retornados pelo *Web Service*. As linhas 2 a 4 apresentam as variáveis que são utilizadas no método.

A linha 6 faz uma busca pela palavra `Confirmacao`, sendo a linha 7 responsável pela busca do caractere `;` a partir da variável `inicio`. O campo `Confirmacao` pode assumir somente dois valores, sendo `S` ou `N`, no qual tem seu valor recuperado a partir da linha 8. A linha 9 recupera o valor de resposta do *Web Service* para a variável `confirmacaoTexto`.

Caso o valor da variável `confirmacaoTexto` seja `"S"`, a variável `confirmacao` recebe `true`, conforme linhas 11 e 12. Após estas verificações, a linha 15 retorna o valor da variável `confirmacao` para quem invocou o método.

Conforme já mencionado sobre a linha 10 da Listagem 6, esta invoca a chamada da próxima tela do sistema, caso usuário tenha informado com sucesso suas credenciais, representada pela classe `Principal.class`. A classe `Principal.class` tem seu *layout* e sua construção apresentados na Figura 12 e na Listagem 7, respectivamente.

Figura 12: Layout Principal.

O *layout* apresentado na Figura 12 representa a interface utilizada pelo usuário para ter acesso às ações que a aplicação possui. Este *layout* pode ser construído com o auxílio de componentes gráficos ou através de construção a partir de código XML, como apresentado na Listagem 7.

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:scrollbars="vertical" android:layout_height="wrap_content">

<ScrollView android:layout_width="fill_parent" android:id="@+id/scrollView1">
    <TableLayout android:id="@+id/tableLayout1" android:layout_width="fill_parent">
        <ImageView android:id="@+id/imageView1" android:src="@drawable/icon_ddetran"/>

        <Button
            android:id="@+id/Veiculo"
            android:layout_width="47dp"
            android:layout_height="47dp"
            android:background="@drawable/veiculo_enabled"/>

        <Button
            android:id="@+id/Condutor"
            android:layout_width="wrap_content"
            android:layout_height="47dp"
            android:background="@drawable/habilitacao_enabled"/>

        <Button android:id="@+id/fechar"
            android:layout_height="47dp"
            android:layout_width="wrap_content"
            android:background="@drawable/sair_enabled"/>

    </TableLayout>
</ScrollView>
</LinearLayout>
```

Listagem 7: Layout Principal.

Através da Figura 12, a qual é resultado da codificação apresentada na Listagem 7, é possível acessar as opções principais da aplicação. Através da linha 3 da Listagem 7 é possível disponibilizar à aplicação a habilidade de ser deslizada para cima ou para baixo através do toque na tela. Esta opção é bastante utilizada caso mais opções surjam e estas não se comportem completamente na tela. A imagem no topo da **Erro! Fonte de referência não encontrada.** é apresentada através da linha 5 da Listagem 7.

Os botões "Veículo", "Habilitação" e "Sair", apresentados na **Erro! Fonte de referência não encontrada.**, são definidos através de códigos da Listagem 7 representados pelas linhas 9 a 13, 15 a 19 e 21 a 24, respectivamente. Os códigos referentes à programação da Figura 12 e Listagem 7 são apresentados na Listagem 8.

```
1  public void onCreate(Bundle savedInstanceState){
2      super.onCreate(savedInstanceState);
3      setContentView(R.layout.main);
4
5      BVeiculo = (Button)findViewById(R.id.Veiculo);
6      BVeiculo.setOnClickListener(new View.OnClickListener(){
7          public void onClick(View v) {
8              Intent myIntent = new Intent(v.getContext(), Veiculo.class);
9              startActivityForResult(myIntent, 0);
10         }
11     });
12
13     BCondutor = (Button)findViewById(R.id.Condutor);
14     BCondutor.setOnClickListener(new View.OnClickListener(){
15         public void onClick(View v) {
16             Intent myIntent = new Intent(v.getContext(), Condutor.class);
17             startActivityForResult(myIntent, 0);
18         }
19     });
20
21     fechar = (Button)findViewById(R.id.fechar);
22     fechar.setOnClickListener(new View.OnClickListener(){
23         public void onClick(View v) {
24             finish();
25         }
26     });
27 }
```

Listagem 8: Programação da classe Principal.

A Listagem 8 apresenta os códigos das ações que os botões da Figura 12 apresentam. A linha 1 define os códigos que são criados/executados na criação da aplicação. Conforme linha 3, este código está ligado ao *layout* `main`, referente à Figura 12.

A linha 5 vincula o botão `BVeiculo` ao botão criado na Figura 12 descrito como “Veículo”. Ao ser clicado, este botão executa a ação descrita dentro dos métodos das linhas 6 e 7. Neste caso, a ação é a chamada da aplicação `Veiculo.class`, sendo criada uma `Intent` na linha 8 e esta sendo iniciada através da Figura 12.

A linha 13 é responsável por vincular o botão `BCondutor` ao botão criado na Figura 12 descrito como “Condutor”. Para este botão, a ação executada ao ser pressionado, conforme linhas 14 e 15, é a criação de uma `Intent` referente à classe `Condutor.class`, sendo invocada logo após, conforme linhas 16 e 17, respectivamente.

Caso o usuário escolha a opção “Sair”, apresentada na Figura 12, a tela é finalizada, conforme linha 24, através da função `finish()`. Por sua vez, a tela apresentada ao usuário através da seleção da opção “Veículo” da Figura 12 pode ser analisada na Figura 13, responsável por apresentar a interface com o usuário, construída através de utilização de componentes gráficos.

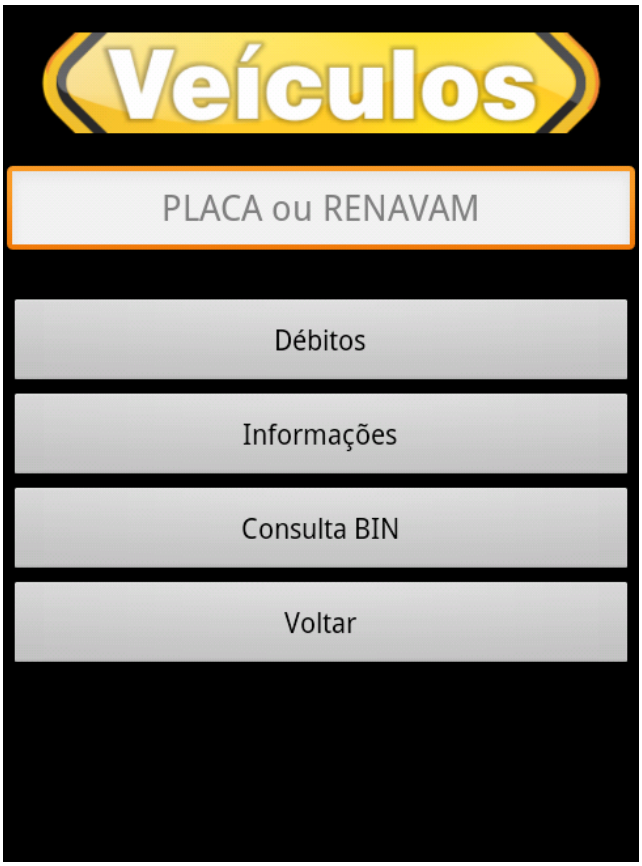

Figura 13: Layout Veículo.

O código apresentado na Listagem 9 se referente à Figura 13. Este código pode ser criado através de componentes gráficos, como feito na Figura 13, ou através da criação de

códigos em XML, apresentados na Listagem 9. Em ambos os casos, o resultado será o mesmo.

```
1  <?xml version="1.0" encoding="utf-8"?>
2
3  <ScrollView android:layout_height="fill_parent"android:id="@+id/scrollView1">
4      <TableLayout android:layout_height="wrap_content" android:id="@+id/tableLayout1">
5          <ImageView android:id="@+id/imageView1" android:src="@drawable/icon_veiculo"/>
6          <EditText android:id="@+id/placaRenavam" android:maxLength="9"
7              android:hint="PLACA ou RENAVAM"/>
8
9          <TextView android:id="@+id/Erro" android:text="" android:textColor="@color/laranja"/>
10
11         <Button android:text="Débitos" android:id="@+id/debitosVeiculo"></Button>
12         <Button android:text="Informações" android:id="@+id/dadosVeiculo"></Button>
13         <Button android:text="Consulta BIN" android:id="@+id/consultaBin"></Button>
14         <Button android:text="Voltar" android:id="@+id/voltar"></Button>
15     </TableLayout>
16 </ScrollView>
```

Listagem 9: Layout Veículo.

Através da tela representada na Figura 13, é possível ter acesso a todas as funcionalidades relacionadas ao veículo. Conforme a Figura 13, as funcionalidades são:

- débitos;
- informações;
- consulta BIN.

Na Listagem 9, é apresentado o código referente à tela de Veículos. Nesta Listagem, a linha 5 é responsável por conter a imagem localizada no topo da Figura 13. O campo para a entrada de dados é referenciado pela linha 6 e 7 da Listagem 9, solicitando a informação da placa ou renavam do veículo a ser pesquisado. Os botões de “Débitos”, “Informações”, “Consulta BIN” e “Voltar” são identificados pelas linhas 11, 12, 13 e 14 da Listagem 9, respectivamente. Já o campo `TextView`, apresentado na linha 9, é responsável por apresentar possíveis problemas na inserção dos dados do veículo, como renavam incorreto. A Listagem 10 apresenta o código que define as ações dos botões apresentados na Figura 13.

```
1   BDebitos = (Button)findViewById(R.id.debitosVeiculo);
2   BDebitos.setOnClickListener(new View.OnClickListener(){
3       public void onClick(View v) {
4           validar();
5
6           if (editRenavamCorreto){
7               Intent myIntent = new Intent(v.getContext(), VeiculoDebitosResultado.class);
8               myIntent.putExtra("renavam", placaRenavamString);
9               myIntent.putExtra("tipo", tipo);
10              startActivityForResult(myIntent, 0);
11          }
12      }
13  });
14
15  BDados = (Button)findViewById(R.id.dadosVeiculo);
16  BDados.setOnClickListener(new View.OnClickListener(){
17      public void onClick(View v) {
18          validar();
19
20          if (editRenavamCorreto){
21              Intent myIntent = new Intent(v.getContext(), VeiculoInformacoesResultado.class);
22              myIntent.putExtra("renavam", placaRenavamString);
23              startActivityForResult(myIntent, 0);
24
25          }
26
27      }
28  });
29
30  BBin = (Button)findViewById(R.id.consultaBin);
31  BBin.setOnClickListener(new View.OnClickListener(){
32      public void onClick(View v) {
33          validar();
34
35          if (editRenavamCorreto){
36              Intent myIntent = new Intent(v.getContext(), VeiculoBinResultado.class);
37              myIntent.putExtra("renavam", placaRenavamString);
38              myIntent.putExtra("tipo", tipo);
39              startActivityForResult(myIntent, 0);
40
41          }
42
43      }
44  });
```

Listagem 10: Código de Veículo.

A Listagem 10 tem seu início na definição do botão “Débitos”, sendo referenciado na implementação como `BDebitos`. A placa ou renavam inserido no `EditText` da Figura 13 é validado através do método `validar()`. Este método é descrito após a descrição da Listagem 10.

Caso a variável `editRenavamCorreto` seja `true`, ou seja, verdadeira, conforme linha 6, é criada uma `Intent` para a apresentação dos débitos do veículo através da classe `VeiculoDebitosResultado.class`. Esta classe recebe dois parâmetros, sendo estes a placa ou renavam e uma variável que identifica qual destas informações o usuário digitou. A placa ou renavam e o tipo são enviados para a classe `VeiculoDebitosResultado.class` através do método `putExtra()` existente na

classe `Intent`, visto nas linhas 8 e 9. O método que invoca esta `Intent` é o apresentado na linha 10.

Os dados referentes a veículo são acessíveis por meio do botão “Informações” da Figura 13. Conforme Listagem 10, a variável do tipo `Button` nomeada `BDados` é mapeada, sendo referência do botão “Informações” da Figura 13. A ação do botão é definida dentro do método das linhas 16 e 17. Caso a variável `editRenavamCorreto` possua valor verdadeiro, é criada uma `Intent` com o parâmetro `VeiculoInformacoesResultado.clas`. Criada a `Intent`, é vinculada a esta o dado de parâmetro que neste caso é a placa ou o renavam, representado pela variável `placaRenavamString`. Para a invocação da tela, é executado o método `startActivityForResult()` passando a `Intent` criada como parâmetro.

As consultas na Base Nacional são representadas pelo botão “Consulta Bin”, conforme Figura 13. Este botão é referenciado pela variável `BBin`, como visto na linha 30 da Listagem 10. A ação do botão é definida nas linhas 31 e 32. Caso seja validado a placa ou renavam informado pelo usuário, é criada uma `Intent` com a classe `VeiculoBinResultado.class` como parâmetro, visto nas linhas 35 e 36. A placa ou renavam e o tipo são vinculados à `Intent`, conforme linhas 37 e 38, e esta é executada para apresentação do resultado ao usuário, visto na linha 39. O código do método `validar()` é descrito na Listagem 11.

```
1  public void validar() {
2          editRenavamCorreto = false;
3
4          erro.setText("");
5          placaRenavamString = placaRenavam.getText().toString();
6
7          if (placaRenavamString.length() == 9){
8              if (placaRenavamString.matches("\\d{9}")){
9                  editRenavamCorreto = true;
10                 tipo = "R";
11             }else{
12                 erro.setText("Número da PLACA ou RENAVAM inválido!"
13                         +"\nFavor informar corretamente!");
14             }
15         }else{
16             if (placaRenavamString.length() == 0){
17                 erro.setText("Favor informar Placa ou Renavam para iniciar pesquisa!");
18             }else{
19                 if (placaRenavamString.matches("\\d+")){
20                     erro.setText("Número da PLACA ou RENAVAM inválido!"
21                             +"\nFavor informar corretamente!");
22                 }else{
23                     if ((placaRenavamString.length() == 5) ||
24                        (placaRenavamString.length() == 6) ||
25                        (placaRenavamString.length() == 7)){
26                         editRenavamCorreto = true;
27                         tipo = "P";
28                     }else{
29                         erro.setText("Número da PLACA ou RENAVAM inválido!"
30                                 +"\nFavor informar corretamente!");
31                     }
32                 }
33             }
34         }
35     }
```

Listagem 11: Validar().

A validação da placa ou renavam, conforme Listagem 11 acontece para verificar se o dado de entrada realmente tem o formato do dado desejado, diminuindo o tráfego e eliminando pesquisas que não retornam resultados.

A variável `placaRenavamString` da linha 5 recebe o valor digitado no `EditText` da Figura 13. Caso o tamanho seja igual a 9 e só existirem caracteres numéricos, vide linha 7 e 8 respectivamente, a variável `editRenavamCorreto` recebe `true` e o tipo `"R"`, certificando que o dado informado referencia um renavam. Caso esta variável seja de tamanho 9 e composta por caracteres alfa numéricos, é apresentada ao usuário uma frase informando que o dado inserido não é uma palca ou renavam, como analisado nas linhas 12 e 13.

Sendo a variável `placaRenavamString` menor que 9, esta só pode se referir a uma placa. A estrutura da placa consiste em ter 2 ou 3 letras seguidas de 4 numerais. Caso possua tamanho zero, visto na linha 16, é apresentada a mensagem de erro ao usuário,

conforme linha 17. Caso a variável possua só valores numéricos, isto é verificado na linha 19 e apresentado o erro nas linhas 20 e 21. Sendo esta verificação anteriormente realizada falsa, caso o dado tenha tamanho 5, 6 ou 7, visto nas linhas 23 a 25, significa que está apto para a pesquisa, fazendo com que a variável `editRenavamCorreto` adquira valor `true` e a variável `tipo` possua valor `"P"`, informando que é referente a uma placa. O tamanho do dado sendo diferente de 5, 6 ou 7, é apresentada ao usuário a informação conforme linhas 29 e 30.

A Listagem 12 e a Figura 14 são responsáveis pelo *layout* ao qual o usuário tem acesso caso seja acionada a opção Habilitação da Figura 12.

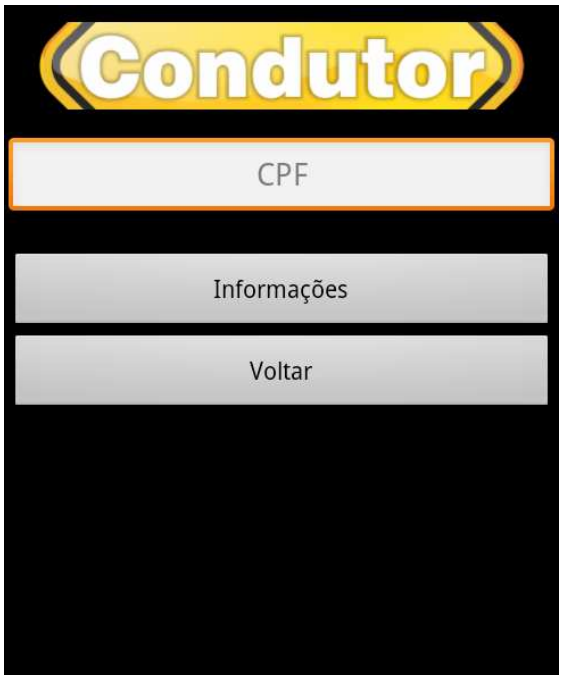

Figura 14: Layout Habilitação.

A Figura 14 representa o *layout* criado com auxílio de componentes gráficos, já a Listagem 12 apresenta o código-fonte referente ao *layout* da **Erro! Fonte de referência não encontrada.**, sendo que a criação do *layout* pode seguir estas duas estruturas.

```
1   <?xml version="1.0" encoding="utf-8"?>
2   <ScrollView android:layout_height="fill_parent" android:id="@+id/scrollView1">
3       <TableLayout android:id="@+id/tableLayout1">
4           <ImageView android:id="@+id/imageView1" android:src="@drawable/icon_condutor"/>
5
6           <EditText android:id="@+id/cpf" android:hint="CPF"
7               android:maxLength="11" android:numeric="integer" />
8
9           <TextView android:id="@+id/Erro" android:textColor="@color/laranja"/>
10          <Button android:text="Informações" android:id="@+id/dadosCondutor"></Button>
11          <Button android:text="Voltar" android:id="@+id/voltar" ></Button>
12      </TableLayout>
13  </ScrollView>
```

Listagem 12: Layout Habilitação.

Conforme a Figura 14 é possível consultar as informações pertinentes a um CPF informado pelo usuário. O código referente a este *layout* é encontrado na Listagem 12. A imagem no topo da Figura 14 é representada pela linha 4 da Listagem 12. A inserção de dados é feita no `EditText` construído nas linhas 6 e 7. Havendo algum erro na informação do CPF pelo usuário, este é apresentado através do `TextView` da linha 9. O botão que realiza a pesquisa das informações está representado pela linha 10 e, caso o usuário opte por sair da opção de “Habilitação”, o botão “Voltar” faz esta ação, desenhado através da linha 11 da Listagem 12.

A Listagem 12 apresenta o código que definem as ações que os botões da Figura 14 executam.

```
1  CPF = (EditText)findViewById(R.id.cpf);
2  erro  = (TextView)findViewById(R.id.Erro);
3
4  BDados = (Button)findViewById(R.id.dadosCondutor);
5  BDados.setOnClickListener(new View.OnClickListener(){
6      public void onClick(View v) {
7
8          if(internetConnection(context)){
9              if(!CPF.getText().toString().equals("")){
10                 if(CPF.getText().toString().length() == 11){
11                     Intent myIntent = new Intent(v.getContext(),
12                                                  CondutorInformacoesResultado.class);
13                     myIntent.putExtra("CPF", CPF.getText().toString());
14                     startActivityForResult(myIntent, 0);
15                 }else{
16                     erro.setText("Informe Usuário(CPF) corretamente!!");
17                 }
18             }else{
19                 erro.setText("Usuário(CPF) é dados obrigatórios!");
20             }
21         }else{
22             erro.setText("Não foi possível estabelecer conexão com o DETRAN-TO!");
23         }
24     }
25 });
26
27 voltar = (Button)findViewById(R.id.voltar);
28 voltar.setOnClickListener(new View.OnClickListener(){
29     public void onClick(View v) {
30         finish();
31     }
32 });
```

Listagem 13: Código de Condutor.

A linha 1 da Listagem 13 é responsável por mapear o `EditText` existente na Figura 14. Através da variável `CPF`, é possível manipular os dados de entrada que o usuário informou. Seguindo a mesma lógica, a variável `erro` é responsável por apresentar possíveis erros ao usuário, através do `TextView Erro` existente no *layout*, conforme linha 9 da Listagem 12.

O botão `dadosCondutor`, existente na linha 10 da Listagem 12, é representado pela variável `BDados`, referenciado na linha 4 da Listagem 13.

Os métodos construídos nas linhas 5 e 6 são referentes a ação do clique do botão. A linha 8 verifica se existe conexão com a Internet, sendo que a falha nesta verificação apresenta ao usuário uma mensagem informando o problema, conforme linha 22. Sendo a verificação da existência de conexão com a Internet bem sucedida, a linha 9 verifica se o usuário informou algo no campo `CPF`. Caso não tenha informado, a linha 19 apresenta o erro ao usuário. Este campo não contendo tamanho 11, a linha 16 informa ao usuário o problema existente, porém, o campo contendo o tamanho 11 desejado, é criada uma `Intent` com o

parâmetro `CondutorInformacoesResultado.class` passando o CPF através do método `putExtra()` e tendo sua inicialização efetivada, conforme linhas 11 a 14.

As próximas seções apresentarão a descrição dos serviços listados nas opções "Veículo" e "Habilitação".

4.2. Consultar Débitos de Veículo

Este serviço é responsável pela pesquisa de débitos de um determinado veículo. Todos os débitos em aberto no Detran-TO são apresentados ao usuário. A Tabela 4 apresenta o dado de entrada para pesquisa de débitos do veículo.

Tabela 4 – Dados enviados pelo Web Service de Consulta Débitos de Veículo.

Item	Elemento	Propriedade
1	PlacaRenavam	String

O item 1 da Tabela 4 apresenta o único dado de entrada do procedimento de consulta de débitos. Este dado pode ser tanto a placa do veículo como o Renavam. A placa do veículo é constituída por duas ou três letras e uma sequencia de 4 números. Caso o dado de entrada seja o Renavam do veículo, este deve ter uma sequencia de 9 dígitos numéricos.

A Tabela 5 apresenta a estrutura de resposta da consulta de débitos do veículo, informando todos os débitos em aberto do veículo, incluindo seus respectivos valores.

Tabela 5 – Dados recebidos pelo *Web Service* de Consulta Débitos de Veículo.

Item	Elemento	Propriedade
1	valorDPVAT	Double
2	valorDPVATAtraso	Double
3	valorInfracoes	Double
4	valorLacre	Double
5	valorLicenciamentoAtual	Double
6	valorNadaConsta	Double
7	valorAtrasoAtual	Double
8	valorAtrasoAtual1	Double
9	valorAtrasoAtual2	Double
10	valorAtrasoAtual3	Double
11	valorAtrasoAtual4	Double
12	valorAtrasoAtual5	Double
13	valorLicenciamentoAtrasadosTotal	Double
14	IndicadorDpvatAtraso	String

15	codBarrasDPVAT	Inteiro
16	codBarrasDPVATAtraso	Inteiro
17	codBarrasDetran	Inteiro
18	Confirmação	String
19	DescricaoErro	String

O universo de dados da Tabela 5, que é enviado como resposta ao aplicativo Android, tem o item 1 representando o valor do seguro DPVAT do ano atual. O item 2 (Tabela 5) representa o total de todos os seguros DPVAT em atraso, ou seja, todos os DPVAT do veículo, exceto o DPVAT do ano atual.

O item 3 da Tabela 5 representa o valor total das infrações que o veículo possui. Estas infrações podem ter sido cometidas de várias formas, tais como multas, apreensão de veículo, recolhimento da CNH, retenção de veículo, entre outros. O item 4 (Tabela 5) é responsável por informar o valor do lacre do veículo, que indica que não houve violação da placa. Este lacre é obrigatório a todos os veículos e só é necessária a lacração uma única vez. O item 5 da Tabela 5 informa o valor do licenciamento do ano atual do veículo. Já o item 6, por sua vez, relata o valor de nada consta para o veículo consultado.

Os itens 7 a 12 (Tabela 5) representam o valor de todos os licenciamentos em atraso para os últimos 5 anos, sendo o item 7 representado pelo ano atual menos 1, o item 8 representado pelo ano atual menos 2 e assim sucessivamente. A soma de todos os atrasos de licenciamento encontra-se localizada no item 13 da Tabela 5. O item 14 (Tabela 5) representa a informação referente a seguros DPVAT não quitados de anos anteriores, sendo *"S"* representando que existe DPVAT sem pagamento para anos anteriores e *"N"* informando que os anos anteriores estão quitados.

Os itens 15 a 17 da Tabela 5 representam os códigos de barra de todos os débitos do veículo, sendo o item 15 responsável pelo valor DPVAT, o item 16 representando o valor total de todos os DPVAT em atraso e o item 17 responsável pelo total de todos os débitos pertinentes ao Detran-TO, como multa, licenciamento, atraso de licenciamento, entre outros.

A confirmação da existência de débitos para o veículo informado é de responsabilidade do item 18 (Tabela 5). Sendo *"S"*, informa que há débitos em aberto e sendo *"N"* informa que todos os débitos foram quitados ou não há lançamento de débitos até a data de consulta.

Caso haja algum erro que impossibilite a verificação dos débitos, este é descrito no item 19 da Tabela 5. Os erros existentes podem ser representados de diversas formas, como a

não existência do veículo na base de dados do Detran-TO, veículo transferido para outro Estado, problema na geração dos débitos, dentre outros.

A próxima seção apresentará a programação realizada para que este método seja executado.

4.2.1. Desenvolvimento do serviço Consultar Débitos de Veículo

Esta seção é responsável por apresentar a descrição da criação do código que realiza a consulta de todos os débitos em aberto de um veículo junto ao Detran-TO. A Figura 15 apresenta o resultado da consulta de débito de um determinado veículo, como visto na primeira opção da Figura 13.

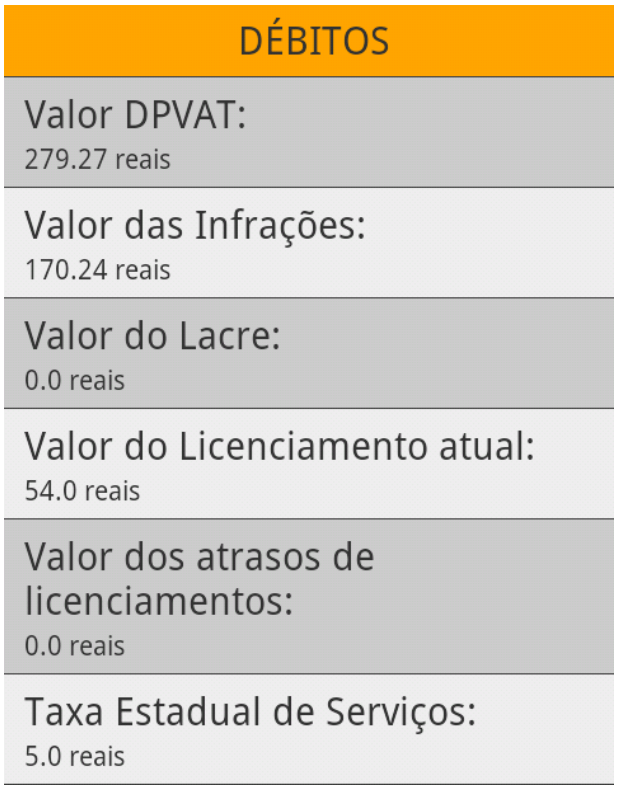

Figura 15: Apresentação de débitos de veículos.

Os valores apresentados na Figura 15 representam todos os débitos em aberto junto ao Detran-TO. A Listagem 14 apresenta o código referente à classe `VeiculoDebitosResultado` que é responsável por apresentar os dados da Figura 15.

```
1  public class VeiculoDebitosResultado extends ListActivity{
2
3      ListView listView;
4      String placaRenavam = "", tipo = "";
5
6      public void onCreate(Bundle savedInstanceState){
7          super.onCreate(savedInstanceState);
8
9          Intent intent = getIntent();
10         placaRenavam = intent.getStringExtra("renavam");
11         tipo = intent.getStringExtra("tipo");
12
13         VeiculoDebitos veiculoSOAP = new VeiculoDebitos(placaRenavam, tipo, this);
14
15         List<VeiculoInfo> veicultoDebitos = new ArrayList<VeiculoInfo>();
16         veicultoDebitos = veiculoSOAP.getVeiculoDebitos();
17
18         setListAdapter(new VeiculoAdapter(this, veicultoDebitos));
19
20     }
21 }
```

Listagem 14: Código da classe VeiculoDebitosResultado.

A classe é definida, na linha 1 da Listagem 14, como uma extensão da classe `ListActivity`. Esta extensão é necessária para apresentação dos dados conforme Figura 15, visto que os dados são apresentados em forma de lista. Não havendo esta extensão, ocorre um erro na chamada desta classe.

Quando esta classe é invocada, o método `onCreate()` é executado. A partir desta chamada, uma variável `intent` é criada e recebe a `Intent` responsável por invocar esta classe, como visto na linha 9. A Intent responsável pela chamada desta classe possui dois parâmetros, sendo eles a placa ou renavam e o tipo, podendo ser analisado através da linha 7 da Listagem 10. Estes parâmetros são recebidos por variáveis, conforme linhas 10 e 11, através do método `intente.getStringExtra()`.

A classe `VeiculoDebitos` é responsável por adquirir os débitos existentes para o veículo, passando como parâmetro a variável referente a placa ou renavam e a variável referente ao tipo, conforme linha 13. Uma lista do tipo `VeiculoInfo` é criada, recebendo todos os débitos do veículo através do método `veiculoSOAP.getVeiculoDebitos()`, analisado nas linhas 15 e 16.

Tendo todos os débitos listados no `ArrayList veiculoDebitos`, é necessário vincular a tela de resultado, Figura 15, aos dados constituídos neste `ArrayList`. Isto é feito através do método `setListAdapter()`, pertencente a classe `ListActivity`. O parâmetro necessário é constituído da classe `VeiculoAdapter` recebendo os valores do

`ArrayList veiculoDebitos`. Após a chamada do método `setListAdapter()`, é apresentado o resultado ao usuário conforme Figura 15. A descrição do código da classe `VeiculoDebitos` é apresentado na Listagem 15.

```
1  public class VeiculoDebitos{
2
3      private static String SOAP_ACTION = "AÇÃO";
4      private static String METHOD_NAME = "SOLICITAR_DEBITOS";
5      private static String NAMESPACE = "NAMESPACE_WEB_SERVICE";
6      private static String URL =      "URL_WEB_SERVICE";
7
8      List<VeiculoInfo> veiculoDebitos = new ArrayList<VeiculoInfo>();
9
10     public VeiculoDebitos(String Renavam, String Tipo, Context context){
11
12         if(internetConnection(context)){
13             SoapObject Request = new SoapObject(NAMESPACE, METHOD_NAME);
14
15             if (Tipo.equals("R")){
16                 Request.addProperty("CodigoRenavam", Renavam);
17                 Request.addProperty("Placa", "");
18             }else{
19                 Request.addProperty("CodigoRenavam", "");
20                 Request.addProperty("Placa", Renavam);
21             }
22             try{
23                 ...
24             }catch(Exception e){
25                 adicionar(e.toString(),"", 1);
26             }
27         }else{
28             adicionar("Não foi possível estabelecer conexão com os servidores do DETRAN-TO!"+
29                     "\nFavor verificar sua conexão com a internet!","", 1);
30         }
31         adicionar("Voltar","", 2);
32     }
33
34     public void adicionar(String Titulo, String Resultado, int Opcao) {
35         VeiculoInfo resultadoConsultaVeiculo = new VeiculoInfo();
36         resultadoConsultaVeiculo.setTitulo(Titulo);
37         resultadoConsultaVeiculo.setResultado(Resultado);
38         resultadoConsultaVeiculo.setOpcao(Opcao);
39         veiculoDebitos.add(resultadoConsultaVeiculo);
40     }
41
42     public List<VeiculoInfo> getVeiculoDebitos(){
43         return veiculoDebitos;
44     }
45 }
```

Listagem 15: Código da classe VeiculoDebitos

As linhas 3 a 6 da Listagem 15 são responsáveis por receberem os dados do *Web Service* que retorna os valores dos débitos do veículo, sendo a ação, método, *namespace* e URL do serviço, respectivamente. A linha 8 constitui da declaração da variável `veiculoDebitos` responsável por ser uma lista de itens do tipo `VeiculoInfo`.

A linha 10 constitui no método construtor da classe, recebendo uma `String` contendo renavam ou placa, uma `String` com o tipo e um parâmetro do tipo `Context`, representando o contexto responsável por invocar esta classe. Havendo conexão com a Internet, linha 12, é criada uma variável do tipo `SoapObject` recebendo como parâmetro o *namespace* e o nome do método do *Web Service* correspondente, visto na linha 13.

A linha 15 verifica se o parâmetro `Tipo` é igual a `“R”`. Sendo igual, é adicionado dois parâmetros no objeto `Request`, através do método `addProperty()`. O primeiro parâmetro tem a responsabilidade de identificar o valor que é passado e o segundo parâmetro possui o valor que se deseja passar. No *Web Service* que é chamado, este identifica os valores através do primeiro parâmetro sendo esta a chave para recuperar o dado desejado, ou seja, caso deseje recuperar o valor do renavam, isto deve ser feito através da identificação `“CodigoRenavam”` ou `“Placa”`.

Caso a linha 15 resulte em falso, as linhas 19 e 20 são responsáveis por adicionar a placa da pesquisa à propriedade `“Placa”`, identificando que a chave passada representa uma placa.

Construído o objeto `Request` e adicionado seus parâmetros, a linha 22 faz a execução da chamada do *Web Service*, analisado na Listagem 16. Caso alguma exceção seja lançada, esta é adicionada à lista `veiculoDebitos` através da função `adicionar()`. Ao final deste processamento, o botão “Voltar” é adicionado no final a lista, possibilitando ao usuário fechar a tela com o demonstrativo de débitos.

O método `adicionar()` tem a função de preencher o `ArrayList veiculoDebitos` com todas as informações referente aos débitos em aberto do veículo. Cada nó deste `ArrayList` é do tipo `VeiculoInfo`, constituído de Título, Resultado e Opção. Após a criação deste objeto `VeiculoInfo`, conforme linhas 35 a 38, este é adicionado ao `ArrayList veiculoDebitos` através da linha 39. Por fim, as linhas 42 a 44 são responsáveis por retornar o `ArrayList veiculoDebitos` contendo todos os débitos do veículo para serem apresentados ao usuário.

```
1  SoapSerializationEnvelope soapEnvelope =
2      new SoapSerializationEnvelope(SoapEnvelope.VER11);
3  soapEnvelope.setOutputSoapObject(Request);
4
5  AndroidHttpTransport aht = new AndroidHttpTransport(URL);
6  aht.call(SOAP_ACTION, soapEnvelope);
7
8  result = (soapEnvelope.getResult()).toString();
9  VeiculoDebitosClasse debitos = new VeiculoDebitosClasse(result);
10
11 adicionar("DÉBITOS","", 1);
12
13 if(debitos.get("Confirmacao").equals("S")){
14     adicionar("Valor DPVAT:",
15         debitos.get("valorDPVAT")+" reais", 3);
16     adicionar("Valor das Infrações:",
17         debitos.get("valorInfracoes")+" reais", 3);
18     adicionar("Valor do Lacre:",
19         debitos.get("valorLacre")+" reais", 3);
20     adicionar("Valor do Licenciamento atual:",
21         debitos.get("valorLicenciamentoAtual")+" reais", 3);
22     adicionar("Valor dos atrasos de licenciamentos:",
23         debitos.get("valorLicenciamentoAtrasadosTotal")+" reais", 3);
24     adicionar("Taxa Estadual de Serviços:",
25         debitos.get("valorNadaConsta")+" reais", 3);
26 }else{
27     if (debitos.get("Confirmacao").equals("N")){
28         adicionar(debitos.get("DescricaoErro"),"", 1);
29     }else{
30         adicionar("Não foi possível estabelecer conexão com os servidores do DETRAN-TO!"
31             +"\nFavor verificar sua conexão com a internet!","", 1);
32     }
33 }
```

Listagem 16: Código TRY da classe VeiculoDebitos.

O trecho de código pertencente a estrutura `try{}` tem seu início na criação do envelope para adquirir os dados dos débitos, analisado nas linhas 1 e 2 da Listagem 16. Este envelope tem como parâmetro de estrutura de resultado o objeto `Request`, criado na linha 13 da Listagem 15, conforme linha 3.

Na linha 5 da Listagem 16, é criado o objeto `aht` correspondente a classe `AndroidHttpTransport` recebendo como parâmetro a `URL` definida na linha 6 da Listagem 15. Este objeto é responsável por efetivamente fazer a comunicação com o *Web Service*, transportando os parâmetros necessários ao *Web Service* e recebendo o resultado conforme o envelope `soapEnvelope`, analisado na linha 6. A linha 8 transforma o resultado adquirido no envelope em `String` e armazena na variável `result`, sendo a linha 9 responsável por criar o objeto `debitos` do tipo `VeiculoDebitosClasse` recebendo a variável `result`. Este objeto contêm todos os débitos extraídos da variável `result`.

A linha 11 da Listagem 16 é responsável por enviar a `String Débitos`, sendo do tipo `"1"` para o método `adicionar()`, visto nas linhas 34 a 40 da Listagem 15. A linha 13

verifica se a resposta do *Web Service* no campo `Confirmacao` foi `“S”`, informando que há débitos em aberto para a placa ou renavam consultado. Sendo a resposta do campo `Confirmacao “N”`, é informado ao usuário o motivo retornado pelo *Web Service* através do campo `DescricaoErro`, visto nas linhas 27 a 29. Caso o campo `Confirmacao` não tenha valor, é informado ao usuário que não houve êxito ao tentar conexão com os servidores do Detran-TO, conforme visto nas linhas 29 a 32 da Listagem 16.

Havendo valores a serem apresentados ao usuário, estes são referentes ao DPVAT, linhas 14 e 15 da Listagem 16; valores das infrações cometidas que estão vinculadas ao veículo, linhas 16 e 17; valor do lacre, linhas 18 e 19; valor do licenciamento do ano corrente, linhas 20 e 21; valor dos licenciamentos que estão abertos e em atraso, linhas 22 e 23; e a taxa estadual de serviço, linhas 24 e 25. Essas informações listadas anteriormente representam todos os débitos que podem estar em aberto junto ao Detran-TO, apresentando ao usuário a informação atual do veículo sobre seus débitos.

A classe `VeiculoDebitosClasse` é apresentada na Listagem 17 a Listagem 21.

```
1  public class VeiculoDebitosClasse{
2
3      private Double valorDPVAT = 0.00, valorDPVATAtraso = 0.00, valorInfracoes = 0.00,
4          valorLacre = 0.00, valorLicenciamentoAtual = 0.00, valorNadaConsta = 0.00,
5          valorAtrasoAtual = 0.00, valorAtrasoAtual1 = 0.00, valorAtrasoAtual2 = 0.00,
6          valorAtrasoAtual3 = 0.00, valorAtrasoAtual4 = 0.00, valorAtrasoAtual5 = 0.00,
7          valorLicenciamentoAtrasadosTotal = 0.00;
8
9      private String IndicadorDpvatAtraso, confirmacao, DescricaoErro;
10
11     private static String KEY_valorDPVAT = "valorDPVAT";
12     private static String KEY_valorDPVATAtraso = "valorDPVATAtraso";
13     private static String KEY_valorInfracoes = "valorInfracoes";
14     private static String KEY_valorLacre = "valorLacre";
15     private static String KEY_valorLicenciamentoAtual = "valorLicenciamentoAtual";
16     private static String KEY_valorNadaConsta = "valorNadaConsta";
17     private static String KEY_valorAtrasoAtual = "ValorAtrasoAtual";
18     private static String KEY_valorAtrasoAtual1 = "ValorAtrasoAtual1";
19     private static String KEY_valorAtrasoAtual2 = "ValorAtrasoAtual2";
20     private static String KEY_valorAtrasoAtual3 = "ValorAtrasoAtual3";
21     private static String KEY_valorAtrasoAtual4 = "ValorAtrasoAtual4";
22     private static String KEY_valorAtrasoAtual5 = "ValorAtrasoAtual5";
23     private static String KEY_valorLicenciamentoAtrasadosTOTAL = "valorLicenciamentoAtrasadosTotal";
24     private static String KEY_IndicadorDpvatAtraso = "IndicadorDpvatAtraso";
25     private static String KEY_confirmacao = "Confirmacao";
26     private static String KEY_DescricaoErro = "DescricaoErro";
27
28     int inicio, fim;
29
30     ...
31 }
```

Listagem 17: Declaração de variáveis da classe VeiculoDebitosClasse.

O código apresentado na Listagem 17 é referente às declarações de variáveis que são utilizadas ao longo do processamento na classe `VeiculoDebitosClasse`. As linhas 3 a 7

são declarações referentes aos valores que podem ser listados para os débitos em aberto do veículo consultado, sendo estes valores do tipo `Double`.

As variáveis declaradas na linha 9 são referentes a informações do tipo `String` que retornam através do *Web Service*. A variável `IndicadorDpvatAtraso` é responsável por retornar `“S”` em caso de existência de DPVAT não quitado em anos anteriores e `“N”` para a não incidência destes débitos. A variável `confirmacao` representa se há débitos em aberto para o veículo consultado, sendo `“S”` para informação da existência e `“N”` para informar a não existência de débitos e a variável `DescricaoErro` reporta uma informação referente aos débitos.

As variáveis das linhas 11 a 26 são as chaves de pesquisa que são utilizadas para serem retornados determinados dados. Os métodos `get()` e `set()`, Listagem 20 e Listagem 21, respectivamente, utilizam essas variáveis para solicitar a alteração ou o retorno dos dados.

```
1  public VeiculoDebitosClasse(String soapEnvelope){
2      inicio = soapEnvelope.indexOf("Confirmacao");
3      fim = soapEnvelope.indexOf(";", inicio);
4      inicio = fim - 1;
5      set("Confirmacao", soapEnvelope.substring(inicio,fim));
6
7      if (get("Confirmacao").equals("S")){
8          inicio = soapEnvelope.indexOf("ValorDpvat");
9          if(inicio != -1){
10             inicio += 11;
11             fim = soapEnvelope.indexOf(";", inicio);
12             fim -= 2;
13             set("valorDPVAT", soapEnvelope.substring(inicio,fim));
14         }
15         inicio = soapEnvelope.indexOf("ValorInfracoes");
16         if(inicio != -1){
17             inicio += 15;
18             fim = soapEnvelope.indexOf(";", inicio);
19             fim -= 2;
20             set("valorInfracoes", soapEnvelope.substring(inicio,fim));
21         }
22         inicio = soapEnvelope.indexOf("ValorLacre");
23         if(inicio != -1){
24             inicio += 11;
25             fim = soapEnvelope.indexOf(";", inicio);
26             fim -= 2;
27             set("valorLacre", soapEnvelope.substring(inicio,fim));
28         }
29         inicio = soapEnvelope.indexOf("ValorLicenciamentoAtual");
30         if(inicio != -1){
31             inicio += 24;
32             fim = soapEnvelope.indexOf(";", inicio);
33             fim -= 2;
34             set("valorLicenciamentoAtual", soapEnvelope.substring(inicio,fim));
35         }
36         int cont = 0, numero = 0;
37         String busca = "";
38
39         ...
40     }
41 }
```

Listagem 18: Código do método construtor da classe VeiculoDebitosClasse.

A Listagem 18 apresenta o código do método construtor da classe `VeiculoDebitosClasse`. A linha 1 representa a declaração deste método recebendo como parâmetro a variável `String` contendo o resultado da consulta via *Web Service*. É necessário encontrar o valor do campo `Confirmacao`, tendo o conhecimento se há ou não débito em aberto para o veículo. Através do método `indexOf()`, linha 2, é feita uma busca com o intuito de buscar da `String "Confirmacao"`, sendo de responsabilidade da variável `inicio` armazenar a posição inicial desta `String`. A variável `fim` armazena a posição da `String ";"` a partir do valor da variável `inicio`. A linha 4 com a variável

`inicio` armazenando a posição `fim – 1` faz com que o método `substring` retorne exatamente o valor desejado, armazenando este valor através do método `set()` passando como parâmetro, além do valor desejado, o valor `“Confirmacao”`.

A linha 7 verifica se há débitos para o veículo consultado, através do método `get(“Confirmacao”)`, o comparado ao dado `“S”`, representante da existência de débito. Havendo sucesso nesta comparação, a mesma lógica aplicada nas linhas 2 a 5 é aplicada para procura dos outros dados retornados pelo *Web Service*, sendo a linha 9 responsável por verificar se o método `indexOf()` da linha 8 encontrou o dado `“ValorDpvat”`, passado como parâmetro. Através da linha 13, é armazenado o valor do DPVAT retornado pelo *Web Service*.

O valor das infrações é adquirido através das linhas 15 a 21. A linha 15 faz a busca da palavra `“ValorInfracoes”` no resultado retornado pelo *Web Service* . Caso tenha encontrado (linha 16), as linhas 17 a 20 verificam o valor da infração e este é armazenado através do método `set()` da linha 20.

O valor do lacre é verificado nas linhas 22 a 28, sendo feita a busca pela palavra `“ValorLacre”` através do método `indexOf()` da linha 22. Este dado sendo encontrado (linha 23), as linhas 22 a 26 são responsáveis por adquirir o valor correto do lacre, sendo a linha 27 responsável por invocar o método `set()`, responsável por armazenar o dado.

As linhas 29 a 35 verificam o valor do licenciamento para o ano corrente, sendo a linha 29 responsável por verificar a existência deste dado. Havendo sucesso na procura deste dado (conforme linha 30), as linhas 31 a 33 realizam a extração exata do dado procurado. A linha 34, por fim, é responsável por guardar este dado através do método `set()`. Por fim, as linhas 36 e 37 inicializam as variáveis que são utilizadas na Listagem 19.

```
1  public VeiculoDebitosClasse(String soapEnvelope){
2          ...
3          while(cont < 6){
4              busca = "ValorAtrasoAtual";
5              if (cont != 0){
6                  busca += String.valueOf(cont);
7                  numero = 1;
8              }
9              inicio = soapEnvelope.indexOf(busca);
10             if(inicio != -1){
11                 inicio += 17 + numero;
12                 fim = soapEnvelope.indexOf(";", inicio);
13                 fim -= 2;
14                 set(busca, soapEnvelope.substring(inicio,fim));
15             }
16
17             cont += 1;
18         }
19         inicio = soapEnvelope.indexOf("IndicadorDpvatAtraso");
20         if(inicio != -1){
21             inicio += 21;
22             fim = soapEnvelope.indexOf(";", inicio);
23
24             if (soapEnvelope.substring(inicio,fim).equals("S")){
25                 set("IndicadorDpvatAtraso", "Sim");
26
27                 //Valor do DPVAT Atraso
28                 inicio = soapEnvelope.indexOf("ValorDpvatAtraso");
29                 if(inicio != -1){
30                     inicio += 17;
31                     fim = soapEnvelope.indexOf(";", inicio);
32                     fim -= 2;
33                     set("valorDPVATAtraso", soapEnvelope.substring(inicio,fim));
34                 }
35             }
36         }
37     }else{
38         inicio = soapEnvelope.indexOf("DescricaoErro");
39         inicio += 14;
40         fim = soapEnvelope.indexOf(";", inicio);
41         set("DescricaoErro", soapEnvelope.substring(inicio,fim));
42     }
```

Listagem 19: Código do método construtor da classe VeiculoDebitosClasse.

A Listagem 19 é responsável por indicar os valores de atraso de licenciamento e a existência de DPVAT em atraso. As linhas 13 a 18 representam um *looping* a ser realizado para recuperação de todos os atrasos de licenciamentos existentes, sendo 6 o número máximo de atrasos cobrados. A linha 19 a 36 são responsáveis por recuperarem o valor do DPVAT atrasado do veículo, caso exista. Esta verificação é realizada por meio do campo `IndicadorDpvatAtraso`, recebendo `“S”` para confirmar que existe DPVAT em atraso ou `“N”` indicando que não há DPVAT em atraso. Havendo a confirmação da existência de

DPVAT em atraso, o valor deste é armazenado através do método `set()` na variável `valorDPVATAtraso`.

Caso a linha 7 da Listagem 18 retorne uma afirmação falsa, as linhas 37 a 42 da Listagem 19 são responsáveis por informar ao usuário qual o erro retornado pelos servidores do Detran-TO através do campo `DescricaoErro`. Esta mensagem é armazenada na variável `DescricaoErro` através do método `set()` da linha 41.

A Listagem 20 é responsável por apresentar a implementação do método `set()` presente na classe `VeiculoDebitosClasse`.

```
1  public void set(Object k, Object v){
2
3      String key = (String) k, value = (String) v;
4
5      if (KEY_valorDPVAT.equals(key))
6          valorDPVAT = Double.valueOf(value.trim()).doubleValue();
7      if (KEY_valorDPVATAtraso.equals(key))
8          valorDPVATAtraso = Double.valueOf(value.trim()).doubleValue();
9      else if (KEY_valorInfracoes.equals(key))
10         valorInfracoes = Double.valueOf(value.trim()).doubleValue();
11     else if (KEY_valorLacre.equals(key))
12         valorLacre = Double.valueOf(value.trim()).doubleValue();
13     else if (KEY_valorLicenciamentoAtual.equals(key))
14         valorLicenciamentoAtual = Double.valueOf(value.trim()).doubleValue();
15     else if (KEY_valorNadaConsta.equals(key))
16         valorNadaConsta = Double.valueOf(value.trim()).doubleValue();
17     else if (KEY_valorAtrasoAtual.equals(key)){
18         valorAtrasoAtual = Double.valueOf(value.trim()).doubleValue();
19         valorLicenciamentoAtrasadosTotal += valorAtrasoAtual;
20     }else if (KEY_valorAtrasoAtual1.equals(key)){
21         valorAtrasoAtual1 = Double.valueOf(value.trim()).doubleValue();
22         valorLicenciamentoAtrasadosTotal += valorAtrasoAtual1;
23     }else if (KEY_valorAtrasoAtual2.equals(key)){
24         valorAtrasoAtual2 = Double.valueOf(value.trim()).doubleValue();
25         valorLicenciamentoAtrasadosTotal += valorAtrasoAtual2;
26     }else if (KEY_valorAtrasoAtual3.equals(key)){
27         valorAtrasoAtual3 = Double.valueOf(value.trim()).doubleValue();
28         valorLicenciamentoAtrasadosTotal += valorAtrasoAtual3;
29     }else if (KEY_valorAtrasoAtual4.equals(key)){
30         valorAtrasoAtual4 = Double.valueOf(value.trim()).doubleValue();
31         valorLicenciamentoAtrasadosTotal += valorAtrasoAtual4;
32     }else if (KEY_valorAtrasoAtual5.equals(key)){
33         valorAtrasoAtual5 = Double.valueOf(value.trim()).doubleValue();
34         valorLicenciamentoAtrasadosTotal += valorAtrasoAtual5;
35     }else if (KEY_IndicadorDpvatAtraso.equals(key))
36         IndicadorDpvatAtraso = value;
37     else if (KEY_confirmacao.equals(key))
38         confirmacao = value;
39     else if (KEY_DescricaoErro.equals(key))
40         DescricaoErro = value;
41 }
```

Listagem 20: Método SET da classe VeiculoDebitosClasse.

O método representado na Listagem 20 verifica o que foi passado no parâmetro `Object k` e no parâmetro `Object v`. A variável `k` se refere às chaves que foram criadas na Listagem 17 e a variável `v` representa o valor desejado a ser armazenado. Ambas as avariáveis são transformadas em tipo `String` conforme linha 3.

É comparada a chave passada como parâmetro e as chaves já existentes declaradas na Listagem 17. Caso o método tenha encontrado a referência da chave, o valor passado como parâmetro na variável `v` é armazenado conforme comparação feita. Sendo o valor a ser armazenado referente a um dado do tipo `Double`, esta transformação é realizada através do método `Double.valueOf()`.

A Listagem 21 apresenta o código referente ao método `get()`, responsável por retornar os valores solicitados.

```
1  public String get(Object k){
2
3      String key = (String) k;
4
5      if (KEY_valorDPVAT.equals(key))
6          return valorDPVAT.toString();
7      if (KEY_valorDPVATAtraso.equals(key))
8          return valorDPVATAtraso.toString();
9      else if (KEY_valorInfracoes.equals(key))
10         return valorInfracoes.toString();
11     else if (KEY_valorLacre.equals(key))
12         return valorLacre.toString();
13     else if (KEY_valorLicenciamentoAtual.equals(key))
14         return valorLicenciamentoAtual.toString();
15     else if (KEY_valorNadaConsta.equals(key))
16         return valorNadaConsta.toString();
17     else if (KEY_valorAtrasoAtual.equals(key))
18         return valorAtrasoAtual.toString();
19     else if (KEY_valorAtrasoAtual1.equals(key))
20         return valorAtrasoAtual1.toString();
21     else if (KEY_valorAtrasoAtual2.equals(key))
22         return valorAtrasoAtual2.toString();
23     else if (KEY_valorAtrasoAtual3.equals(key))
24         return valorAtrasoAtual3.toString();
25     else if (KEY_valorAtrasoAtual4.equals(key))
26         return valorAtrasoAtual4.toString();
27     else if (KEY_valorAtrasoAtual5.equals(key))
28         return valorAtrasoAtual5.toString();
29     else if (KEY_valorLicenciamentoAtrasadosTOTAL.equals(key))
30         return valorLicenciamentoAtrasadosTotal.toString();
31     else if (KEY_IndicadorDpvatAtraso.equals(key))
32         return IndicadorDpvatAtraso;
33     else if (KEY_confirmacao.equals(key))
34         return confirmacao;
35     else if (KEY_DescricaoErro.equals(key))
36         return DescricaoErro;
37     return null;
38 }
```

Listagem 21: Método GET da classe VeiculoDebitosClasse.

O método apresentado na Listagem 21 recebe uma variável do tipo `Object`. Este objeto é transformado em uma `String`, representando a chave a ser pesquisada, conforme linha 3. As linhas a partir dessa atribuição são responsáveis por verificar qual chave foi enviada no parâmetro, sendo que a partir do sucesso da checagem, o dado referente a solicitação seja retornado.

4.3. Consultar Informações de Veículo no Tocantins

Este serviço tem a responsabilidade de listar todas as informações referentes ao veículo consultado na base de dados do Detran-TO. A Tabela 4 apresenta o dado de entrada deste serviço, sendo este a chave de pesquisa para a busca dos dados.

A Tabela 6 apresenta os dados retornados da base de dados do Detran-TO, contendo as informações do veículo.

Tabela 6 – Dados recebidos pelo *Web Service* de Consultar Informações de Veículo no Tocantins.

Item	Elemento	Propriedade
1	Placa	String
2	Renavam	String
3	Chassi	String
4	Cor	String
5	Marca	Inteiro
6	Cidade	String
7	CodIBGE	Inteiro
8	Nome	String
9	Recadastrado	String
10	OrigemCadastro	String
11	PlacaAnterior	String
12	CPFCNPJAnterior	String
13	NomeAnterior	String
14	RecadastradoAnterior	Short
15	NomeAlienante	String
16	TipoAlienacao	String
17	TipoRestricao	String
18	DescricaoBaixa	String
19	TipoVeiculo	String
20	Combustivel	String
21	Categoria	String
22	Especie	String
23	Nacionalidade	String
24	Carroceria	String
25	FormaLicenciamento	String
26	NumeroMotor	String
27	Situacao	String
28	SituacaoLacre	String
29	AnoFabr	Inteiro
30	AnoModelo	Inteiro
31	CodigoCor	Inteiro
32	CodigoMarca	Inteiro

33	CodigoCidade	Inteiro
34	TipoPessoaProprietario	Inteiro
35	CGCCPF	Inteiro
36	ExercLicenciamento	Inteiro
37	CodigoTipoVeiculo	Inteiro
38	CodigoCombustivel	Inteiro
39	CodigoCategoria	Inteiro
40	Lugares	Inteiro
41	NumeroEixos	Inteiro
42	PotenciaVeiculo	Inteiro
43	DataAquisicao	Data
44	DataUltimaAtualizacao	Data
45	DataLicenciamento	Data
46	Erro	String

Os dados retornados da base de dados do Detran-TO, representados pela Tabela 6, tem o item 1 responsável por conter a placa atual do veículo, o item 2 contendo o número Renavam, o item 3 contendo o chassi do veículo e o item 4 responsável por conter a descrição da cor do veículo consultado.

O item 5 da Tabela 6 fica responsável por conter a descrição da marca do veículo. O item 6 tem a informação da cidade onde o veículo se encontra atualmente cadastrado. Já o código IBGE da cidade onde o veículo está cadastrado se encontra no item 7 da mesma Tabela.

O nome do proprietário atual do veículo é encontrado no item 8 da Tabela 6. O item 9 é responsável por conter a descrição do recadastro do veículo nos Órgãos vinculados ao Detran-TO. Já o item 10 representa a origem do cadastro, podendo assumir valores que informam ao usuário que o veículo foi cadastrado pelo sistema atual do Detran-TO, sistema antigo ou por algum arquivo de lote que foi processado no sistema atual do Detran-TO.

O item 11 da Tabela 6 apresenta a placa anterior do veículo, caso exista. O item 12 representa o CPF ou CNPJ do antigo proprietário, assim como o item 13 apresenta o nome deste antigo proprietário. O item 14 da Tabela 6 informa se o veículo foi recadastrado anteriormente em outro sistema que não seja o atual utilizado pelos órgãos do Detran-TO. Caso o veículo consultado possua alguma alienação, os itens 15 e 16 apresentam os dados do alienante vinculado ao veículo, contendo no item 15 o nome do alienante e o item 16 o tipo de alienação feita para este veículo, podendo tais tipos variar em fiduciário, arrendamento mercantil ou penhor.

O item 17 da Tabela 6 apresenta o tipo de restrição vinculada ao veículo. Esta restrição pode possuir vários valores, sendo alguns deles referente a ordens judiciárias, baixa definitiva,

suspensão de tráfego, entre outros. A descrição do motivo da baixa do veículo é informada no item 18. O item 19 representa o tipo de veículo a qual as informações da Tabela 6 pertencem, como automóvel, motocicleta, triciclo, trator, caminhão, etc. O item 20 descreve o combustível referente ao veículo, podendo variar em gasolina, álcool, bicombustível e gás.

A categoria do veículo é descrita no item 21 da Tabela 6, podendo assumir alguns valores como particular, arrendado, frotista e locadora. O item 22 apresenta a espécie do veículo, normalmente atribuída informação de que o veículo tem finalidade para passageiro. A nacionalidade do veículo é informada no item 23, sendo nacional ou internacional. O item 24 da Tabela 6 representa a informação se há ou não carroceria vinculada ao veículo. No item 25 é relatada a forma de licenciamento que há para o veículo. O item 26 apresenta o número do motor, o item 27 a situação atual do veículo e o item 28 a situação do lacre.

O item 29 da Tabela 6 tem a responsabilidade de informar o ano de fabricação do veículo. O ano do modelo do veículo é informado no item 30. Os códigos da cor, marca/modelo e cidade são apresentados nos itens 31, 32 e 33, respectivamente. O item 34 informa o tipo de proprietário que está associado ao veículo, sendo pessoa física ou jurídica. Já o item 35 é responsável por conter o número do documento.

A informação referente ao último ano de licenciamento para o veículo trafegar em vias públicas consta no item 36 da Tabela 6. O código da marca/modelo do veículo está no item 37, assim como o código de combustível e categoria estão nos itens 38 e 39, respectivamente. O número de lugares do veículo se encontra no item 40, assim como o número de eixos informados no item 41 e a potência do veículo constando no item 42.

A data de aquisição do veículo, data de última atualização na base de dados do Detran-TO e a data do licenciamento do veículo estão nos itens 43, 44 e 45 da Tabela 6, em sequência.

Caso haja algum erro que impossibilite o acesso aos dados na base de dados do Detran-TO, este é relatado no item 46 da Tabela 6, podendo assumir vários valores como não constar na base de dados do Detran-TO, possuir alguma restrição de busca ou veículo ainda não ter o processo de transferência para o Tocantins concluído.

4.3.1. Desenvolvimento do serviço Informações de Veículo no Tocantins

Esta seção é responsável por apresentar a descrição da criação do código que realiza a consulta de todas as informações de um veículo junto ao Detran-TO. A Figura 16 apresenta o

resultado da consulta de informações de um determinado veículo, como visto na segunda opção da Figura 16.

Figura 16: Apresentação de informações de veículos.

Os dados apresentados na Figura 16 são referentes às informações de um determinado veículo junto ao Detran-TO, sendo apresentadas em tempo real. A Listagem 22 apresenta o código referente à classe `VeiculoInformacoesResultado` que é responsável por apresentar os dados da Figura 16.

```
1  package ws.detran.asmx.VeiculoInformacoes;
2
3  public class VeiculoInformacoesResultado extends ListActivity{
4
5      ListView listView;
6      String placaRenavam = "";
7
8      public void onCreate(Bundle savedInstanceState){
9          super.onCreate(savedInstanceState);
10
11         Intent intent = getIntent();
12         placaRenavam = intent.getStringExtra("renavam");
13
14
15         VeiculoInformacoes veiculoSOAP = new VeiculoInformacoes(placaRenavam, this);
16
17         List<VeiculoInfo> veicultoDados = new ArrayList<VeiculoInfo>();
18         veicultoDados = veiculoSOAP.getVeiculoDados();
19
20         setListAdapter(new VeiculoAdapter(this, veicultoDados));
21     }
22 }
```

Listagem 22: Código da classe VeiculoInformacoesResultado.

A Listagem 22 representa a classe que apresenta os dados para o usuário. Esta classe recebe os dados enviados pela tela anterior através das linhas 11 e 12, recebendo, neste caso, a placa ou renavam enviado através do método `intente.getStringExtra("renavam")` e armazenando o dado na variável `placaRenavam`.

A variável `placaRenavam` é passada como parâmetro no construtor da classe `VeiculoInformacoes`, conforme linha 15. Esta classe é a responsável por conectar com os servidores do Detran-TO, recebendo o resultado.

A linha 17 caracteriza por criação do objeto `veiculoDados` do tipo `List<VeiculoInfo>`, sendo este objeto responsável por receber o resultado da consulta ao servidor do Detran-TO, através da linha 18.

O objeto `veiculoDados` devidamente preenchido, é necessário informar ao sistema Android que cada item deste objeto está vinculado a uma estrutura XML criada. Esta associação é realizada através do método `setListAdapter()` da linha 20, passando como parâmetro a classe `VeiculoAdapter` que realiza esta associação.

A Listagem 23 apresenta o código da classe `VeiculoInformacoes`.

```
1  public class VeiculoInformacoes{
2
3      private static String SOAP_ACTION = "AÇÃO";
4      private static String METHOD_NAME = "DADOS_DO_VEICULO";
5      private static String NAMESPACE = "NAMESPACE_WEB_SERVICE";
6      private static String URL =      "URL_WEB_SERVICE";
7
8      List<VeiculoInfo> veicultoDados = new ArrayList<VeiculoInfo>();
9
10     public VeiculoInformacoes(String Renavam, Context context){
11
12         if(internetConnection(context)){
13
14             try{
15                 ...
16             }catch(Exception e){
17                 adicionar(e.toString(),"", 1);
18             }
19         }else{
20             adicionar("Não foi possível estabelecer conexão com os servidores do DETRAN-TO!\n"+
21                       "Favor verificar sua conexão com a internet!","", 1);
22         }
23         adicionar("Voltar","", 2);
24     }
25
26     public void adicionar(String Titulo, String Resultado, int Opcao) {
27         VeiculoInfo resultadoConsultaVeiculo = new VeiculoInfo();
28         resultadoConsultaVeiculo.setTitulo(Titulo);
29         resultadoConsultaVeiculo.setResultado(Resultado);
30         resultadoConsultaVeiculo.setOpcao(Opcao);
31         veicultoDados.add(resultadoConsultaVeiculo);
32     }
33
34     public List<VeiculoInfo> getVeiculoDados(){
35         return veicultoDados;
36     }
37 }
```

Listagem 23: Código da classe VeiculoInformacoes.

A classe representada na Listagem 23 tem suas atribuições bem próximas às atribuições da Listagem 16, sendo a definição dos parâmetros do *Web Service* definida nas linhas 3 a 6 e o construtor definido nas linhas 10 a 24.

Na linha 12 é verificada a conexão com a Internet e na linha 14 é definida a instrução `try{}`. Havendo alguma exceção lançada, as linhas 16 a 18 exibem para o usuário o erro ocorrido. Não havendo conexão com a Internet, uma mensagem é exibida ao usuário através das linhas 19 a 22. Ao final, o botão “Voltar” é adicionado a lista de dados.

O método `adicionar()`, linhas 26 a 32, recebe o título da informação, o resultado e a opção. Com estes dados, é criado um objeto do tipo `VeiculoInfo` e estes dados recebidos como parâmetro são integrados a este objeto, sendo ao final da integração adicionado à lista `veiculoDados`, conforme linha 31. As linhas 34 a 36 são responsáveis por retornar a lista de dados do veículo contendo todas as suas características.

Para efetuar a comunicação com o *Web Service* e fazer a recuperação de dados, apresentando estes ao usuário, é apresentada a instrução `try{}`, conforme Listagem 24.

```
1  try{
2      SoapObject Request = new SoapObject(NAMESPACE, METHOD_NAME);
3      Request.addProperty("ChavePesquisa", Renavam);
4
5      SoapSerializationEnvelope soapEnvelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
6      soapEnvelope.setOutputSoapObject(Request);
7
8      AndroidHttpTransport aht = new AndroidHttpTransport(URL);
9      aht.call(SOAP_ACTION, soapEnvelope);
10
11     result = (soapEnvelope.getResult()).toString();
12
13     VeiculoInformacoesClasse informacoes = new VeiculoInformacoesClasse(result);
14
15     adicionar("DADOS GERAIS de VEÍCULO","", 1);
16     if(informacoes.get("Erro") == "N"){
17
18         adicionar(informacoes.get("DescricaoRouboRestricao"), informacoes.get("DescricaoRouboDetalhada"), 4);
19         adicionar("Placa:",informacoes.get("Placa"), 3);
20         adicionar("Renavam:", informacoes.get("Renavam"), 3);
21         adicionar("Placa Anterior:",informacoes.get("PlacaAnterior"), 3);
22         adicionar("Categoria:",informacoes.get("CodigoCategoria")+" - "+informacoes.get("Categoria"), 3);
23         adicionar("Espécie:",informacoes.get("CodigoEspecie")+" - "+informacoes.get("Especie"), 3);
24         adicionar("Cor:",informacoes.get("CodigoCor")+" - "+informacoes.get("Cor"), 3);
25         adicionar("Marca/Modelo:",informacoes.get("CodigoMarca")+" - "+informacoes.get("Marca"), 3);
26         adicionar("Fabricação/Modelo:",informacoes.get("AnoFabr")+'/'+informacoes.get("AnoModelo"), 3);
27         adicionar("Nome do Proprietário:",informacoes.get("Nome"), 3);
28         adicionar("Licenciado:",informacoes.get("FormaLicenciamento"), 3);
29         adicionar("Adquirido em:",informacoes.get("DataAquisicao"), 3);
30         adicionar("Situação em:",informacoes.get("DescricaoBaixa"), 3);
31         adicionar("Restrição à Venda:",informacoes.get("TipoAlienacao"), 3);
32         adicionar("CARACTERÍSTICA DO VEÍCULO","", 1);
33         adicionar("Chassi:",informacoes.get("Chassi"), 3);
34         adicionar("Número do Motor:",informacoes.get("NumeroMotor"), 3);
35         adicionar("Número do Lacre:",informacoes.get("NumeroLacreFormatado"), 3);
36         adicionar("Quantidade de Eixos:",informacoes.get("QuantidadeEixos"), 3);
37 }else{
38         adicionar("Veículo não cadastrado na base de dados do DETRAN/TO","", 1);
39 }
```

Listagem 24: Código da classe VeiculoInformacoes.

As linhas 2 e 3 da Listagem 24 apresenta a criação do objeto `Request` referente ao *Web Service* representado pelas variáveis `NAMESPACE` e `METHOD_NAME`. A placa ou renavam é anexado ao objeto através do método `addProperty()` referenciado pela palavra `ChavePesquisa`.

Os dados retornados pelo *Web Service* são guardados no objeto `soapEnvelope`, criado na linha 5. Este objeto tem sua estrutura de retorno baseada no objeto `Request`, conforme linha 6, através do método `setOutputSoapObject()`.

O objeto que conecta com o *Web Service* através da Internet é o `aht`, referente à classe `AndroidHttpTransport`, recebendo a variável `URL` e sendo acionado através da linha 9 pelo método `call()`, informando a ação, contida na variável `SOAP_ACTION`, e recebendo o retorno, através do `soapEnvelope`.

O objeto `soapEnvelope` é transformado em `String` e armazenado na variável `result`, conforme linha 11. A linha 13 cria o objeto `informacoes` do tipo `VeiculoInformacoesClasse`, recebendo como parâmetro a variável `result`.

A linha 15 usa o método `adicionar()` para começar a estruturar os resultados que são apresentados ao usuário. A primeira apresentação de dados é feita a partir desta linha. Caso não tenha acontecido nenhum erro do retorno das informações do veículo, verificado na linha 16, as linhas 18 a 36 adicionam os dados retornados à lista de dados `veiculoDados` com o auxílio do método `adicionar()`. Caso exista algum erro retornado pelo *Web Service*, as linhas 37 a 39 informam ao usuário que o veículo não existe na base de dados do Detran-TO.

A Listagem 25 apresenta o início da criação da classe `VeiculoInformacoesClasse`.

```
1  public class VeiculoInformacoesClasse{
2
3      private int TipoTransacaoSNG, TipoRestricaoRoubo, Sequencia, AnoFabr, AnoModelo,
4          CodigoCor, CodigoMarca, CodigoCidade, TipoPessoaProprietario, CGCCPF,
5          Titularidade, CodigoTipoAlienacao, TitularidadeAlienante, MotivoBaixa,
6          ExercLicenciamento, CodigoTipoVeiculo, CodigoCombustivel, CodigoCategoria,
7          idProcessoAtualizacao, idPessoaProprietarioAnterior, idPessoaProprietario,
8          idCidadePais, IndicadorSituacao, ConformidadeVeiculo, idEndereco,
9          PendenteLacre, NumeroLacre;
10     private String DataInclusaoSNG, TipoRestricaoSNG, NomeFinanciadoSNG, NomeAgenteSNG,
11         DescricaoRouboRestricao, DescricaoRouboDetalhada, Erro, Placa, Renavam, Chassi,
12         Cor, Marca, Cidade, CodIBGE, Nome, Recadastrado, OrigemCadastro, PlacaAnterior,
13         EspecieAbreviada, CarroceriaAbreviada, TipoVeiculoAbreviado, CombustivelAbreviado,
14         SituacaoLacre, NumeroLacreFormatado,  UsuarioInclusao, DataAquisicao;
15     private double CapacidadeCarga, PBTVeiculo, CMTVeiculo;
16     private Date DataUltimaAtualizacao, DataLicenciamento;
17
18     ...
19 }
```

Listagem 25: Código da classe VeiculoInformacoesClasse.

A classe `VeiculoInformacoesClasse` tem suas variáveis apresentadas conforme Listagem 25. Essas variáveis são divididas em `int`, `String`, `double` e `Date`. As variáveis apresentadas armazenam todos os dados retornados pelo *Web Service*.

A Listagem 26 dá continuação à apresentação do código da classe `VeiculoInformacoesClasse`.

```
1  public class VeiculoInformacoesClasse{
2      ...
3      private static String KEY_Erro = "Erro";
4      private static String KEY_Placa = "Placa";
5      private static String KEY_Renavam = "Renavam";
6      private static String KEY_Chassi = "Chassi";
7      private static String KEY_Cor = "Cor";
8      private static String KEY_Marca = "Marca";
9      private static String KEY_Cidade = "Cidade";
10     private static String KEY_CodIBGE = "CodIBGE";
11     private static String KEY_Nome = "Nome";
12     private static String KEY_TipoVeiculo = "TipoVeiculo";
13     private static String KEY_Combustivel = "Combustivel";
14     private static String KEY_Categoria = "Categoria";
15     private static String KEY_PrimeiroRegistro = "PrimeiroRegistro";
16     private static String KEY_Especie = "Especie";
17     private static String KEY_Situacao = "Situacao";
18     private static String KEY_CategoriaAbreviada = "CategoriaAbreviada";
19     private static String KEY_EspecieAbreviada = "EspecieAbreviada";
20     private static String KEY_AnoFabr = "AnoFabr";
21     private static String KEY_AnoModelo = "AnoModelo";
22     private static String KEY_CodigoCor = "CodigoCor";
23     private static String KEY_CodigoMarca = "CodigoMarca";
24     private static String KEY_CodigoCidade = "CodigoCidade";
25     private static String KEY_Aquisicao = "Aquisicao";
26     private static String KEY_CodigoEspecie = "CodigoEspecie";
27     private static String KEY_CategoriaDPVAT = "CategoriaDPVAT";
28     private static String KEY_PlacaFinal = "PlacaFinal";
29     private static String KEY_IndicadorSituacao = "IndicadorSituacao";
30     private static String KEY_ConformidadeVeiculo = "ConformidadeVeiculo";
31     private static String KEY_TipoTransacaoSNG = "TipoTransacaoSNG";
32     private static String KEY_NomeFinanciadoSNG = "NomeFinanciadoSNG";
33     private static String KEY_TipoRestricaoSNG = "TipoRestricaoSNG";
34     private static String KEY_DataInclusaoSNG = "DataInclusaoSNG";
35     private static String KEY_NomeAgenteSNG = "NomeAgenteSNG";
36
37     ...
38 }
```

Listagem 26: Código da classe VeiculoInformacoesClasse.

As variáveis que são chave para pesquisa de dados estão representadas na Listagem 26. Através do valor destas chaves é possível solicitar o valor referente a determinado dado. Um exemplo é a solicitação da placa do veículo, sendo necessário apenas enviar a palavra `"Placa"` ao método `get()`, descrito na Listagem 29, para que o valor da variável `Placa`, criada conforme Listagem 25, seja retornado.

A Listagem 27 apresenta o método construtor da classe `VeiculoInformacoesClasse`.

```
1  public class VeiculoInformacoesClasse{
2      ...
3      public VeiculoInformacoesClasse(String soapEnvelope){
4          if (soapEnvelope.indexOf("Placa") != -1){
5              set("Erro", "N");
6              inicio = soapEnvelope.indexOf("Placa");
7              if(inicio != -1){
8                  inicio = soapEnvelope.indexOf("=", inicio);
9                  inicio += 1;
10                 fim = soapEnvelope.indexOf(";", inicio);
11                 set("Placa", soapEnvelope.substring(inicio,fim));
12             }
13
14             inicio = soapEnvelope.indexOf("Renavam");
15             if(inicio != -1){
16                 inicio = soapEnvelope.indexOf("=", inicio);
17                 inicio += 1;
18                 fim = soapEnvelope.indexOf(";", inicio);
19                 set("Renavam", soapEnvelope.substring(inicio,fim));
20             }
21
22             inicio = soapEnvelope.indexOf("TipoVeiculo", inicio);
23             if(inicio != -1){
24                 inicio = soapEnvelope.indexOf("=", inicio);
25                 inicio += 1;
26                 fim = soapEnvelope.indexOf(";", inicio);
27                 set("TipoVeiculo", soapEnvelope.substring(inicio,fim));
28             }
29
30             inicio = soapEnvelope.indexOf("CodigoCategoria");
31             if(inicio != -1){
32                 inicio = soapEnvelope.indexOf("=", inicio);
33                 inicio += 1;
34                 fim = soapEnvelope.indexOf(";", inicio);
35                 set("CodigoCategoria", soapEnvelope.substring(inicio,fim));
36             }
37             ...
38         }else{
39             set("Erro", "S");
40         }
41     }
42 }
```

Listagem 27: Código da classe VeiculoInformacoesClasse.

O código da Listagem 27 apresenta a busca pelas informações existentes nas variáveis `soapEnvelope`. A busca é feita conforme os valores apresentados nas chaves, vide Listagem 26, sendo feito para cada chave existente. Na Listagem 27 são apresentadas somente algumas recuperações de informação, visto que para as demais recuperações somente é necessária a mudança do parâmetro `indexOf()`.

A linha 4 verifica a existência da informação “Placa”, visto que se esta informação não existe, resulta no não sucesso da verificação da existência do veículo na base de dados do Detran-TO. Esta verificação sendo verdadeira, as linhas 6 a 12 buscam o dado, utilizando o método `set()` para efetuar o armazenamento do dado na variável `Placa`. O mesmo procedimento é feito para o renavam, linhas 14 a 20, para o tipo de veículo, conforme linhas 22 a 28 e para o código da categoria, visto nas linhas 30 a 36. Caso a checagem da linha 4 não tenha sucesso, a variável `Erro` recebe o valor “S” através do método `set()`.

O método `set()` da classe `VeiculoInformacoesClasse` é apresentado na Listagem 28.

```
1  public class VeiculoInformacoesClasse{
2      ...
3
4      public void set(Object k, Object v){
5
6          String key = (String) k;
7          String value = (String) v;
8
9          if (KEY_Erro.equals(key))
10             Erro = value;
11         else if (KEY_Placa.equals(key))
12             Placa = value;
13         else if (KEY_Renavam.equals(key))
14             Renavam = value;
15         else if (KEY_PlacaAnterior.equals(key))
16             PlacaAnterior = value;
17         else if (KEY_CodigoTipoVeiculo.equals(key))
18             CodigoTipoVeiculo = Integer.parseInt(value);
19         else if (KEY_TipoVeiculo.equals(key))
20             TipoVeiculo = value;
21         else if (KEY_CodigoCategoria.equals(key))
22             CodigoCategoria = Integer.parseInt(value);
23         ...
24     }
25 }
```

Listagem 28: Método SET da classe VeiculoInformacoesClasse.

Os parâmetros recebidos pelo método `set()` são a chave, `Object k`, e o valor, `Object v`, conforme visto na linha 4 da Listagem 28. Os parâmetros são transformados em `String` através das linhas 6 e 7 e armazenados nas variáveis `key` e `value`, respectivamente.

As estruturas `if else if` apresentadas nas linhas 9 a 23 são responsáveis por verificar se as chaves mencionadas na Listagem 26 são iguais à variável `key`, responsável por conter a informação de qual campo se deseja guardar o dado. Um exemplo é a primeira verificação, linhas 9 e 10, sendo responsáveis por verificar se a chave `KEY_Erro` é igual ao dado presente na variável `key`. Essa verificação sendo verdadeira, a variável `Erro` recebe o valor apresentado na variável `value`. Este procedimento é repetido para todas as chaves existentes na classe.

A Listagem 29 apresenta o código referente ao método `get()` da classe `VeiculoInformacoesClasse`.

```
1  public class VeiculoInformacoesClasse{
2      ...
3
4      public String get(Object k){
5
6          String key = (String) k;
7
8          if (KEY_Erro.equals(key))
9              return Erro;
10         else if (KEY_Placa.equals(key))
11             return Placa.toString();
12         else if (KEY_Renavam.equals(key))
13             return Renavam.toString();
14         else if (KEY_PlacaAnterior.equals(key))
15             return PlacaAnterior.toString();
16         else if (KEY_TipoVeiculo.equals(key))
17             return TipoVeiculo.toString();
18         else if (KEY_Categoria.equals(key))
19             return Categoria.toString();
20         else if (KEY_Especie.equals(key))
21             return Especie.toString();
22         else if (KEY_Cor.equals(key))
23             return Cor.toString();
24         else if (KEY_Marca.equals(key))
25             return Marca.toString();
26         ...
27
28         return null;
29     }
30 }
```

Listagem 29: Método GET da classe VeiculoInformacoesClasse.

O método `get()` apresentado na Listagem 29 recebe um `object k` responsável por conter o dado a ser retornado. Este dado é transformado em uma `String`, representado pela variável `key`, conforme mostrado na linha 6.

A variável `key` é comparada às chaves descritas na Listagem 26, sendo que caso exista igualdade deve ser retornado o valor referente àquela chave. Um exemplo que pode ser analisado é referente às linhas 8 e 9, onde caso o valor existente na variável `key` seja igual a chave `KEY_Erro` é retornado a variável Erro declarada conforme Listagem 25. Esta rotina é aplicada a todas as chaves existentes. Não sendo encontrada nenhuma chave correspondente é retornado o valor `null`.

4.4. Consultar Informações de Veículo na Base Nacional

O serviço a ser relatado nesta seção se refere à consulta de informações do veículo na Base Nacional. Este procedimento se faz imprescindível pela necessidade de consulta de informações de veículos de outros Estados, já que no Estado do Tocantins trafegam veículos não exclusivamente da região, sendo os veículos do tipo caminhão os mais encontrados com esta característica, devido à localização geográfica estratégica que o Estado possui.

A nomenclatura dos campos retornados pela Base Nacional é diferente da nomenclatura utilizada nos sistemas do Detran-TO. A padronização da nomenclatura com base na estrutura da Base Nacional foi cogitada, mas devida a falta de conhecimento e/ou o risco de possíveis alterações na nomenclatura por parte da Base Nacional, optou-se pela utilização da nomenclatura da Base Nacional neste serviço, mantendo o padrão da origem dos dados do remetente.

O dado de entrada desta consulta pode ser analisado na Tabela 4, sendo a Tabela 7 responsável por informar todos os dados retornados como resposta da solicitação.

Tabela 7 – Dados recebidos pelo *Web Service* de Consultar Informacoes de Veículo na Base Nacional.

Item	Elemento	Propriedade
1	IdTipoVeiculo	Inteiro
2	IdMarca	Inteiro
3	IdEspecie	Inteiro
4	IdTipoCarroceria	Inteiro
5	IdCor	Inteiro
6	AnoModelo	Inteiro

7	AnoFabricacao	Inteiro
8	Potencia	Inteiro
9	IdCombustivel	Inteiro
10	Placa	String
11	Renavam	String
12	Chassi	String
13	SituacaoVeiculo	String
14	NomeLocal	String
15	PlacaUF	String
16	NomeTipoVeiculo	String
17	NomeMarca	String
18	NomeEspecie	String
19	NomeTipoCarroceria	String
20	NomeCor	String
21	NomeCombustivel	String
22	IdentMotor	String
23	Nacional	String
24	CapacidadeCarga	Inteiro
25	TipoDocumentoProprietario	String
26	DocumentoProprietario	Inteiro
27	CapacidadePassageiros	Inteiro
28	NomeRestricao	String
29	QuantidadeEixos	Inteiro
30	TipoDocumentoFatur	String
31	DocumentoFatur	String
32	UFFatur	String
33	ComunicacaoVenda	String
34	TextoComunicacaoVenda	String
35	PendenciaEmissao	String
36	TextoPendenciaEmissao	String
37	DataUltimaAtualizacao	Data
38	Erro	Inteiro
39	DescricaoErro	String

O item 1 da Tabela 7 representa o código do tipo de veículo consultado. Todos os códigos utilizados nos sistemas internos do Detran-TO são padronizados de acordo com os códigos existentes na Base Nacional. Sendo assim, caso um veículo possui código de tipo de veículo igual a 6 retornado, este é identificado como “automóvel” tanto no sistema da Base Nacional como nos sistemas do Detran-TO.

O item 2 da Tabela 7 se refere ao código da marca/modelo pertencente ao veículo. O código da espécie é relatado no item 3 da mesma Tabela, sendo o código da carroceria do veículo representado pelo item 4. A identificação do código da cor se encontra no item 5.

O ano referente ao modelo do veículo consta no item 6 da Tabela 7, assim como o ano de fabricação é referenciado pelo item 7. A potência do veículo é vinculada no item 8, bem como o item 9 representa o código do tipo de combustível encontrado no veículo.

A placa atual do veículo é representada pelo item 10 da Tabela 7 e o item 11 representante do Renavam. O chassi é apresentado no item 12 e a situação do veículo é relatada no item 13, sendo alguns exemplos de situação do veículo as informações “Veículo em circulação”, “Consta baixa definitiva de veículo” e “Impedimento jurídico em processamento”.

O item 14 da Tabela 7 apresenta o nome atual da cidade que está vinculado o veículo consultado e o item 15 representa o Estado, sendo este identificado por 2 letras, representando a sigla do mesmo.

O item 16 da Tabela 7 informa a descrição do tipo de veículo. Já o item 17 da mesma Tabela é responsável pela descrição da marca/modelo, o item 18 pela descrição da espécie do veículo, item 19 pela descrição do tipo de carroceria, item 20 pela descrição da cor utilizada e o item 21 apresenta a descrição do tipo de combustível utilizado.

A identificação do número do motor é relatada no item 22 da Tabela 7. A informação referente à nacionalidade do veículo, sendo nacional ou internacional, está descrita no item 23 da mesma Tabela. O número referente à capacidade de carga que possui o veículo é informado no item 24, sendo Tonelada a unidade de medida utilizada.

O item 25 da Tabela 7 é responsável por informar o tipo de documento que o proprietário do veículo possui e o item 26 informa o número deste documento do proprietário. A quantidade máxima de passageiros é relatada no item 27 da mesma Tabela.

Caso haja alguma restrição que impossibilite a circulação do veículo pelas vias públicas, esta é informada no item 28 da Tabela 7. A quantidade de eixos existentes no veículo é apresentada no item 29 da mesma Tabela.

Sobre o faturamento do veículo, ou seja, informações referentes à compra do veículo, o item 30 da Tabela 7 informa o tipo de documento pertencente ao responsável, o item 31 informa o documento do responsável e o item 32 informa o Estado onde houve tal faturamento do veículo.

Havendo comunicado de venda associado a este veículo, esta informação é apresentada no item 33 da Tabela 7 e no item 34 consta o texto referente ao comunicado de venda existente. O item 35 referencia a existência de pendências de emissão de documento, caso existam. O item 36 da mesma Tabela informa a descrição da pendência existente.

O item 37 da Tabela 7 informa a data da última atualização do veículo processada na Base Nacional. Por fim, os itens 38 e 39 da mesma Tabela são representantes de possíveis erros que possam existir, sendo o item 38 responsável por informar um número de erro interno da Base Nacional e o item 39 a descrição deste número, apresentando ao usuário uma explicação do erro existente.

4.4.1. Desenvolvimento do serviço Informações de Veículo na Base Nacional

Esta seção é responsável por apresentar a descrição da criação do código que fará a consulta de todas as informações de um veículo junto à Base Nacional. A Figura 17 apresenta o resultado da consulta de informações de um determinado veículo através da terceira opção da Figura 17.

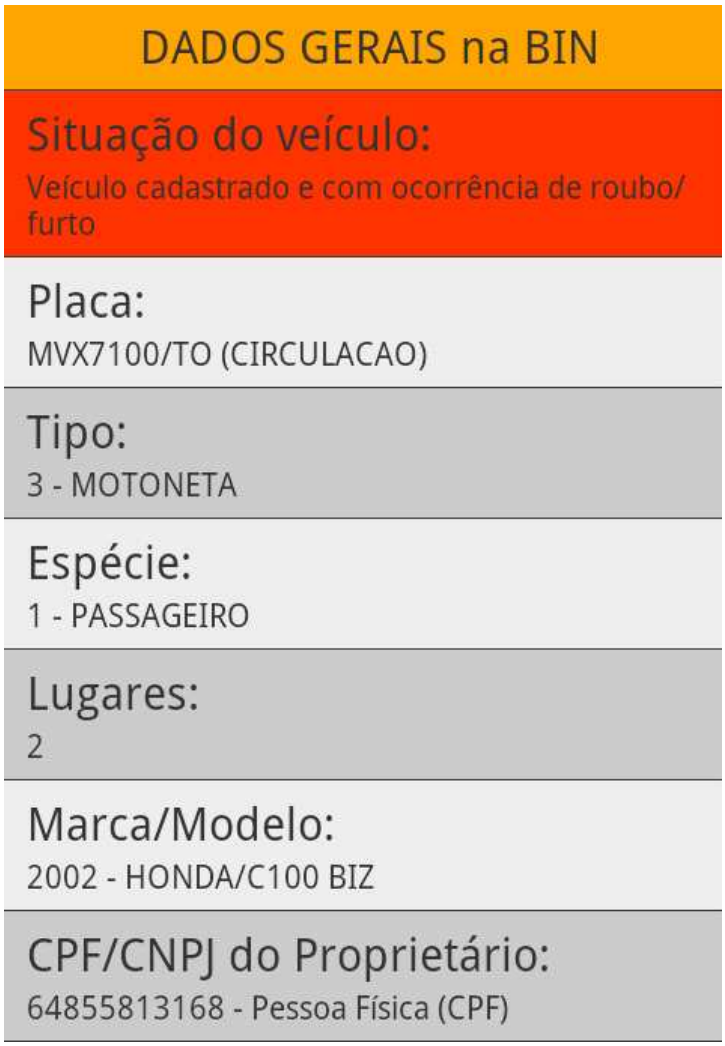

Figura 17: Apresentação de informações de veículos na BIN.

O resultado apresentado na Figura 17 é referente a um veículo que teve seus dados solicitados junto à Base Nacional. Esta apresentação de dados é feita através da classe `VeiculoBinResultado`, apresentada na Listagem 30.

```
1  public class VeiculoBinResultado extends ListActivity{
2
3      ListView listView;
4      String placaRenavam = "", tipo = "";
5
6      public void onCreate(Bundle savedInstanceState){
7          super.onCreate(savedInstanceState);
8
9          Intent intent = getIntent();
10         placaRenavam = intent.getStringExtra("renavam");
11         tipo = intent.getStringExtra("tipo");
12
13         VeiculoBin veiculoSOAP = new VeiculoBin(placaRenavam, tipo, this);
14
15         List<VeiculoInfo> veicultoBin = new ArrayList<VeiculoInfo>();
16         veicultoBin = veiculoSOAP.getVeiculoBin();
17
18         setListAdapter(new VeiculoAdapter(this, veicultoBin));
19     }
20 }
```

Listagem 30: Código da classe VeiculoBinResultado.

A classe `VeiculoBinResultado`, conforme Listagem 30, faz a apresentação de dados retornados da Base Nacional. Esta classe é uma extensão da classe `ListActivity`, já que há a necessidade de apresentação de dados em forma de lista, conforme apresentado na Figura 17.

As linhas 3 e 4 apresentam as variáveis que são utilizadas na classe. Já as linhas 6 e 7 apresentam o método `onCreate()` utilizado para a criação da aplicação. Esta classe necessita da placa ou renavam do veículo e o tipo deste dado para fazer a consulta na Base Nacional. Estes dados são recuperados a partir das linhas 9 a 11, enviados pela classe que invocou a classe da Listagem 30. A linha 13 faz a criação do objeto `veiculoSOAP` do tipo `VeiculoBin` enviando as variáveis `placaRenavam` e `tipo` que são utilizados no processamento do método construtor da classe.

A linha 15 define o objeto `veiculoBin` do tipo `List<VeiculoInfo>` que recebe todos os dados retornados da Base Nacional para apresentação ao usuário, sendo a linha 16 responsável por recuperar estes dados através do método `getVeiculoBin()` do objeto `veiculoSOAP`.

Após a montagem do objeto `veiculoBin` com todos os dados solicitados à Base Nacional, é necessário a vinculação deste objeto com a estrutura XML criada para apresentação dos dados. Esta associação é feita através do método `setListAdapter()`, linha 18, passando como parâmetro a classe `VeiculoAdapter` que realiza esta associação.

A Listagem 31 apresenta o código da classe `VeiculoBin`.

```
1  public class VeiculoBin{
2      private static String SOAP_ACTION = "AÇÃO";
3      private static String METHOD_NAME = "DADOS_DO_VEICULO_BIN";
4      private static String NAMESPACE = "NAMESPACE_WEB_SERVICE";
5      private static String URL =      "URL_WEB_SERVICE";
6
7      List<VeiculoInfo> veicultoBin = new ArrayList<VeiculoInfo>();
8
9      public VeiculoBin(String PlacaRenavam, String Tipo, Context context){
10
11         if(internetConnection(context)){
12             try{
13                 ...
14             }catch(Exception e){
15                 adicionar(e.toString(),"", 1);
16             }
17         }else{
18             adicionar("Não foi possível estabelecer conexão com os servidores do DETRAN-TO!"+
19                       "\nFavor verificar sua conexão com a internet!","", 1);
20         }
21         adicionar("Voltar","botao", 2);
22     }
23
24     public void adicionar(String Titulo, String Resultado, int Opcao) {
25         VeiculoInfo resultadoConsultaVeiculo = new VeiculoInfo();
26         resultadoConsultaVeiculo.setTitulo(Titulo);
27         resultadoConsultaVeiculo.setResultado(Resultado);
28         resultadoConsultaVeiculo.setOpcao(Opcao);
29         veicultoBin.add(resultadoConsultaVeiculo);
30     }
31
32     public List<VeiculoInfo> getVeiculoBin(){
33         return veicultoBin;
34     }
35 }
```

Listagem 31: Código da classe VeiculoBin.

A Listagem 31 possui a mesma estrutura lógica da classe apresentada na Listagem 23. A diferença entre as Listagens se encontra na linha 9 da Listagem 31, sendo passado o parâmetro `Tipo` a mais, responsável por identificar se o parâmetro `PlacaRenavam` se refere a uma placa ou renavam.

A estrutura `try{}`, apresentada na linha 12, terá o seu código exibido na Listagem 32.

```
1  SoapObject Request = new SoapObject(NAMESPACE, METHOD_NAME);
2  Request.addProperty("ChavePesquisa", PlacaRenavam);
3
4  SoapSerializationEnvelope soapEnvelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
5  soapEnvelope.dotNet = true;
6  soapEnvelope.setOutputSoapObject(Request);
7
8  AndroidHttpTransport aht = new AndroidHttpTransport(URL);
9  aht.call(SOAP_ACTION, soapEnvelope);
10
11 result = (soapEnvelope.getResult()).toString();
12 VeiculoBinClasse veiculoBin = new VeiculoBinClasse(result);
13
14 adicionar("DADOS GERAIS na BIN","", 1);
15 if(veiculoBin.get("Erro").equals("N")){
16
17     adicionar("Situação do veículo:", veiculoBin.get("SituacaoExecucao"), 4);
18     adicionar("Placa:", veiculoBin.get("Placa")+"/"+veiculoBin.get("PlacaUF")
19         +" ("+veiculoBin.get("SituacaoVeiculo")+")", 3);
20     adicionar("Tipo:",veiculoBin.get("IdTipoVeiculo")+" - "+veiculoBin.get("NomeTipoVeiculo"), 3);
21     adicionar("Espécie:",veiculoBin.get("IdEspecie")+" - "+veiculoBin.get("NomeEspecie"), 3);
22     adicionar("Lugares:",veiculoBin.get("CapacidadePassageiros"), 3);
23     adicionar("Marca/Modelo:",veiculoBin.get("IdMarca")+" - "+veiculoBin.get("NomeMarca"), 3);
24     adicionar("Fabricação/Modelo:",veiculoBin.get("AnoFabricacao")+"/"+veiculoBin.get("AnoModelo"), 3);
25     adicionar("Combustível:",veiculoBin.get("IdCombustivel")+" - "+veiculoBin.get("NomeCombustivel"), 3);
26     adicionar("Renavam:",veiculoBin.get("Renavam"), 3);
27     adicionar("Chassi:",veiculoBin.get("Chassi")+" ("+veiculoBin.get("NomeRemarcacao")+")", 3);
28     adicionar("Cor:",veiculoBin.get("IdCor")+" - "+veiculoBin.get("NomeCor"), 3);
29     adicionar("Potência/Cilindradas:",veiculoBin.get("Potencia")+"/"+veiculoBin.get("Cilindrada"), 3);
30     adicionar("Restrição:",veiculoBin.get("NomeRestricao"), 3);
31     adicionar("UF Faturamento:",veiculoBin.get("UFFatur"), 3);
32     adicionar("CNPJ Faturamento:",veiculoBin.get("DocumentoFatur"), 3);
33 }else{
34     if (veiculoBin.get("Erro").equals("S")){
35         adicionar(veiculoBin.get("DescricaoErro"),"", 1);
36     }else{
37         adicionar("Não foi possível estabelecer conexão com os servidores do DETRAN-TO!"+
38                 "\nFavor verificar sua conexão com a internet!","", 1);
39     }
40 }
```

Listagem 32: Código da classe VeiculoBin.

As linhas 1 a 15 da Listagem 32 apresentam as mesmas informações e códigos presentes na estrutura `try{}` da Listagem 24, possuindo a mesma lógica das linhas 1 a 16.

As linhas 17 a 32 utilizam o método `adicionar()`, apresentado na Listagem 31, linhas 24 a 30, para armazenamento dos dados na lista de dados `VeiculoBin`.

Havendo algum erro presente na recuperação dos dados, este erro é relatado ao usuário por meio das linhas 34 e 35. Não havendo uma descrição oriunda dos servidores do Detran-TO, uma mensagem informando a existência de problemas na conexão com o Órgão é apresentada conforme linhas 36 a 38.

A classe responsável por extrair os dados da estrutura retornada pelo Detran-TO é chamada de `VeiculoBinClasse`. Esta classe tem a mesma estrutura da classe `VeiculoInformacoesClasse`, apresentada Listagem 26, Listagem 27, Listagem 28, Listagem 29 e Listagem 29, finalizando a descrição do código deste método de consulta de informações na Base Nacional.

4.5. Consultar Informações de Condutor

O serviço de consulta de informações do condutor visa buscar todas as informações a cerca de pessoas físicas, relatando à aplicação Android todas suas informações atualizadas.

O dado de entrada a ser enviado é o CPF do condutor a qual se tenha interesse em consultar. Realizada a solicitação de consulta de um condutor, a Tabela 8 apresenta o resultado da consulta solicitada.

Tabela 8 – Dados recebidos pelo *Web Service* de Consultar Informações de Condutor.

Item	Elemento	Propriedade
1	IdPessoa	Inteiro
2	DescricaoRetorno	String
3	CPF	Inteiro
4	Nome	String
5	NomeMae	String
6	NomePai	String
7	DocumentoIdentidadeNumero	String
8	DocumentoIdentidadeOrgaoEmissor	String
9	DocumentoIdentidadeUF	String
10	DataNascimento	Data
11	LocalNascimento	String
12	UfNascimento	String
13	Sexo	String
14	Logradouro	String
15	Complemento	String
16	Bairro	String
17	LocalEndereco	String
18	UFEndereco	String
19	CEP	String
20	NomeCiretran	String
21	DataInclusao	Data
22	NumeroRENACH	Inteiro
23	DataValidadeCNH	Data
24	CategoriaCNH	String
25	DataPrimeiraHabilitacao	Data
26	Erro	String

O item 1 da Tabela 8 apresenta um número sequencial atribuído ao condutor no momento do seu cadastro efetivado nos sistemas do Detran-TO, sendo este número único e chave para a identificação de todas as informações referentes ao condutor no sistema a nível

de Banco de Dados, utilizado internamente para representar chaves estrangeiras no Banco de Dados.

O item 2 da Tabela 8 representa a descrição do código de retorno, informando que, neste caso, o retorno tem destino externo ao Órgão Detran-TO, por ser uma aplicação Android.

Os itens de 3 a 12 da Tabela 8 contém os dados referente à Carteira de Identidade do condutor, sendo o item 3 responsável por conter o CPF, o item 4 o nome do condutor, o item 5 o nome da mãe, o item 6 o nome do pai, o item 7 o número do documento de identidade, o item 8 o órgão emissor responsável e o item 9 o Estado de origem desta Carteira de Identidade. O item 10 da Tabela 8 se refere à data de nascimento do condutor, bem como o item 11 o local de nascimento e o item 12 o Estado de nascimento do mesmo.

Por sua vez, o sexo do condutor é apresentado no item 13 da Tabela 8.

Os itens apresentados nos números 14 a 19 da Tabela 8 se referem à localidade atual do condutor. O item 14 informa o logradouro, o item 15 a descrição do complemento do endereço, o item 16 a referencia ao bairro, o item 17 a cidade, o item 18 o Estado e o item 19 apresenta o CEP do endereço do condutor.

A CIRETRAN (Circunscrição Regional de Trânsito) do condutor é apresentada no item 20 da Tabela 8, sendo este o estabelecimento de atendimento ao público em geral de responsabilidade do Detran-TO presente em quase todas as cidades do Estado. Por sua vez, a data da inclusão do cadastro do condutor está presente no item 21.

O número Renach do condutor é descrito no item 22 da Tabela 8. Este número é gerado para representar todos os processos referentes ao condutor.

A data de validade da CNH do condutor é expressa no item 23 da Tabela 8, assim como a categoria desta CNH é exibida no item 24. Já o item 25 apresenta a data da primeira habilitação vinculada ao condutor.

Por fim, o item 26 da Tabela 8 informa possíveis erros existentes referentes ao CPF consultado, sendo alguns exemplos de erros o não sucesso na consulta com base no CPF informado, o CPF não pertencer a um condutor credenciado no Tocantins, o condutor não pertencer mais ao Estado, entre outros.

4.5.1. Desenvolvimento do serviço Informações de Condutor

Esta seção apresenta o código de implementação do serviço que retorna os dados de um determinado contribuinte, presentes na base de dados do Detran-TO. A Figura 18 apresenta a consulta de informações de um determinado condutor.

Figura 18: Apresentação de informações de condutor.

A apresentação dos dados da Figura 18 é de responsabilidade da classe `CondutorInformacoesResultado`, apresentada na Listagem 33.

```
1  public class CondutorInformacoesResultado extends ListActivity{
2
3      ListView listView;
4      String CPF = "";
5
6      public void onCreate(Bundle savedInstanceState){
7          super.onCreate(savedInstanceState);
8
9          Intent intent = getIntent();
10         CPF = intent.getStringExtra("CPF");
11
12         CondutorInformacoes veiculoSOAP = new CondutorInformacoes(CPF, this);
13
14         List<CondutorInfo> condutorDados = new ArrayList<CondutorInfo>();
15         condutorDados = veiculoSOAP.getCondutorDados();
16
17         setListAdapter(new CondutorAdapter(this, condutorDados));
18     }
19 }
```

Listagem 33: Código da classe CondutorInformacoesResultado.

A classe `CondutorInformacoesResultado` é uma extensão da classe `ListActivity`, já que está vinculada a uma apresentação de dados em forma de lista, visto na linha 1 da Listagem 33. As variáveis são definidas nas linhas 3 e 4 e o método `onCreate()`, responsável pela criação da aplicação, é definido nas linhas 6 e 7. Esta classe utiliza o CPF do condutor, sendo esta informação adquirida através das linhas 9 e 10, recuperando o dado `"CPF"` através do método `getStringExtra()` da `Intent` que invocou esta classe.

A linha 12 da Listagem 33 cria o objeto `veiculoSOAP` e seu método construtor recebe a variável CPF, recuperando todos os dados pertinentes na base de dados do Detran-TO. As linhas 14 e 15 são responsáveis por criar o objeto `condutorDados` e alimentá-lo com os dados recuperados da base de dados do Detran-TO, através do método `getCondutorDados()`.

A apresentação dos dados é de responsabilidade da linha 17, através do método `setListerAdpater()`, recebendo a classe `CondutorAdapter` e a lista de dados `condutorDados` a ser apresentada.

A Listagem 34 apresenta o código da classe `CondutorInformacoes`.

```
1  public class CondutorInformacoes {
2
3      private static String SOAP_ACTION = "AÇÃO";
4      private static String METHOD_NAME = "DADOS_DO_VEICULO";
5      private static String NAMESPACE = "NAMESPACE_WEB_SERVICE";
6      private static String URL =      "URL_WEB_SERVICE";
7
8      List<CondutorInfo> condutorDados = new ArrayList<CondutorInfo>();
9
10     public CondutorInformacoes(String CPF, Context context){
11         if(internetConnection(context)){
12             try{
13                 ...
14             }catch(Exception e){
15                 adicionar(e.toString(),"", 1);
16             }
17         }else{
18             adicionar("Não foi possível estabelecer conexão com os servidores do DETRAN-TO!\n"+
19                       "Favor verificar sua conexão com a internet!","", 1);
20         }
21         adicionar("Voltar","", 2);
22     }
23
24     public void adicionar(String Titulo, String Resultado, int Opcao) {
25         CondutorInfo resultadoConsultaCondutor = new CondutorInfo();
26         resultadoConsultaCondutor.setTitulo(Titulo);
27         resultadoConsultaCondutor.setResultado(Resultado);
28         resultadoConsultaCondutor.setOpcao(Opcao);
29         condutorDados.add(resultadoConsultaCondutor);
30     }
31
32     public List<CondutorInfo> getCondutorDados(){
33         return condutorDados;
34     }
35 }
```

Listagem 34: Código da classe CondutorInformacoes.

A Listagem 34 possui a mesma estrutura lógica da classe apresentada na Listagem 24. A diferença entre as Listagens está presente na linha 10, sendo passado o parâmetro CPF ao invés do parâmetro Renavam. Outra mudança se refere ao método `adicionar()` localizado na linha 24, sendo a lista de dados referente a classe `CondutorInfo`.

A estrutura `try{}`, apresentada na linha 12, tem seu código exibido na Listagem 35Listagem 32.

```
1  try{
2      SoapObject Request = new SoapObject(NAMESPACE, METHOD_NAME);
3      Request.addProperty("CPF", CPF);
4
5      SoapSerializationEnvelope soapEnvelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
6      soapEnvelope.setOutputSoapObject(Request);
7
8      AndroidHttpTransport aht = new AndroidHttpTransport(URL);
9      aht.call(SOAP_ACTION, soapEnvelope);
10
11     result = (soapEnvelope.getResult()).toString();
12
13     CondutorInformacoesClasse informacoes = new CondutorInformacoesClasse(result);
14
15     adicionar("DADOS GERAIS de CONDUTOR","", 1);
16     if(!informacoes.get("IdPessoa").equals("0")){
17         adicionar("Nome:",informacoes.get("Nome"), 3);
18         adicionar("CPF:",informacoes.get("CPF"), 3);
19         adicionar("Sexo:",informacoes.get("Sexo"), 3);
20         adicionar("Nome Mãe:",informacoes.get("NomeMae"), 3);
21         adicionar("Nome Pai:",informacoes.get("NomePai"), 3);
22         adicionar("Data Nascimento:",informacoes.get("dataNascimento"), 3);
23         adicionar("Ciretran:",informacoes.get("NomeCiretran"), 3);
24         adicionar("Cadastro DETRAN:",informacoes.get("DataInclusao"), 3);
25         adicionar("Recadastrado DETRAN:",informacoes.get("Recadastrado"), 3);
26         adicionar("Dados Habilitação","", 1);
27         adicionar("Nº RENACH:",informacoes.get("NumeroRENACH"), 3);
28         adicionar("Validade CNH:",informacoes.get("DataValidadeCNH"), 3);
29         adicionar("Categoria Atual:",informacoes.get("CategoriaCNH"), 3);
30         adicionar("Nº Registro:",informacoes.get("NumeroRegistro1"), 3);
31         adicionar("Nº CNH:",informacoes.get("NumeroFormularioCNH"), 3);
32         adicionar("Data 1ª Habilitação:",informacoes.get("DataPrimeiraHabilitacao"), 3);
33     }else{
34         adicionar(informacoes.get("DescErro"),"", 1);
35     }
36 }
```

Listagem 35: Código da classe CondutorInformacoes.

O trecho de código apresentado nas linhas 2 a 15 tem suas características semelhantes às apresentadas na Listagem 24 mudando apenas o parâmetro adicionado na linha 3, referente ao CPF do condutor que se deseja consultar.

Caso a consulta pelo condutor tenha tido sucesso, a linha 16 apresenta a afirmação verdadeira, sendo o `"IdPessoa"` diferente de `"0"`. Após esta verificação, as linhas 17 a 32 utilizam o método `adicionar()` passando como parâmetro os dados retornados pelos servidores do Detran-TO. Caso a consulta pelo condutor não tenha sucesso, a linha 34 apresenta o erro existente para o usuário.

A Listagem 36 apresenta as variáveis que são utilizadas na classe `CondutorInformacoesVeiculo`.

```
1  public class CondutorInformacoesClasse {
2
3      int IdPessoa = 0, Sexo = 0, CodigoLocalNascimento = 0, CodigoLocalEndereco = 0;
4
5      String DataPrimeiraHabilitacao = "", NumeroFormularioCNH = "", NumeroRegistro1 = "",
6          CategoriaCNH = "", DataValidadeCNH = "", NumeroRENACH = "", DescErro = "",
7          Recadastrado = "", DataInclusao = "", NomeCiretran = "", CPF = "",
8          DescricaoRetorno = "", Nome = "", NomeMae = "", NomePai = "",
9          DocumentoIdentidadeNumero = "", DocumentoIdentidadeOrgaoEmissor = "",
10         DocumentoIdentidadeUF = "", dataNascimento = "", LocalNascimento = "",
11         UfNascimento = "", Logradouro = "", Complemento = "", Bairro = "",
12         LocalEndereco = "", UFEndereco = "", CEP = "";
13 }
```

Listagem 36: Código da classe CondutorInformacoesClasse.

As variáveis utilizadas na classe são definidas conforme linhas 3 a 12 da Listagem 36 e são separadas em inteiros e `String`. Estas variáveis armazenam os dados retornados pelo *Web Service* que consulta os dados do condutor.

A Listagem 37 apresenta o código referente às chaves que são utilizadas para recuperação dos dados referentes ao condutor.

```
1  public class CondutorInformacoesClasse {
2      ...
3
4      private static String KEY_IdPessoa = "IdPessoa";
5      private static String KEY_DescricaoRetorno = "DescricaoRetorno";
6      private static String KEY_CPF = "CPF";
7      private static String KEY_Nome = "Nome";
8      private static String KEY_NomeMae = "NomeMae";
9      private static String KEY_NomePai = "NomePai";
10     private static String KEY_DocumentoIdentidadeNumero = "DocumentoIdentidadeNumero";
11     private static String KEY_DocumentoIdentidadeOrgaoEmissor = "DocumentoIdentidadeOrgaoEmissor";
12     private static String KEY_DocumentoIdentidadeUF = "DocumentoIdentidadeUF";
13     private static String KEY_Sexo = "Sexo";
14     private static String KEY_dataNascimento = "dataNascimento";
15     private static String KEY_LocalNascimento = "LocalNascimento";
16     private static String KEY_UfNascimento = "UfNascimento";
17     private static String KEY_Logradouro = "Logradouro";
18     private static String KEY_Complemento = "Complemento";
19     private static String KEY_Bairro = "Bairro";
20     private static String KEY_LocalEndereco = "LocalEndereco";
21     private static String KEY_UFEndereco = "UFEndereco";
22     private static String KEY_CEP = "CEP";
23     private static String KEY_NomeCiretran = "NomeCiretran";
24     private static String KEY_DataInclusao = "DataInclusao";
25     private static String KEY_Recadastrado = "Recadastrado";
26     private static String KEY_DescErro = "DescErro";
27     private static String KEY_NumeroRENACH = "NumeroRENACH";
28     private static String KEY_DataValidadeCNH = "DataValidadeCNH";
29     private static String KEY_CategoriaCNH = "CategoriaCNH";
30     private static String KEY_NumeroRegistro1 = "NumeroRegistro1";
31     private static String KEY_NumeroFormularioCNH = "NumeroFormularioCNH";
32     private static String KEY_DataPrimeiraHabilitacao = "DataPrimeiraHabilitacao";
33
34     ...
35 }
```

Listagem 37: Código da classe CondutorInformacoesClasse.

As variáveis que são chaves das pesquisas para recuperação de dados são definidas na Listagem 37. Caso seja solicitada a recuperação do CPF, é passado para o parâmetro `get()` a palavra `“CPF”` que é comparado às chaves existentes na classe. Sendo encontrada a chave referente ao CPF, o método retorna o valor solicitado.

A Listagem 38 apresenta o método construtor da classe `CondutorInformacoesClasse`.

```
1  public class CondutorInformacoesClasse {
2      ...
3      public CondutorInformacoesClasse(String soapEnvelope){
4          public CondutorInformacoesClasse(String soapEnvelope){
5              inicio = soapEnvelope.indexOf("IdPessoa");
6              if(inicio != -1){
7                  inicio = soapEnvelope.indexOf("=", inicio);
8                  inicio += 1;
9                  fim = soapEnvelope.indexOf(";", inicio);
10                 set("IdPessoa", soapEnvelope.substring(inicio,fim));
11             }
12
13             inicio = soapEnvelope.indexOf("CPF");
14             if(inicio != -1){
15                 inicio = soapEnvelope.indexOf("=", inicio);
16                 inicio += 1;
17                 fim = soapEnvelope.indexOf(";", inicio);
18                 set("CPF", soapEnvelope.substring(inicio,fim));
19             }
20
21             inicio = soapEnvelope.indexOf("Nome");
22             if(inicio != -1){
23                 inicio = soapEnvelope.indexOf("=", inicio);
24                 inicio += 1;
25                 fim = soapEnvelope.indexOf(";", inicio);
26                 set("Nome", soapEnvelope.substring(inicio,fim));
27             }
28
29             inicio = soapEnvelope.indexOf("NomeMae");
30             if(inicio != -1){
31                 inicio = soapEnvelope.indexOf("=", inicio);
32                 inicio += 1;
33                 fim = soapEnvelope.indexOf(";", inicio);
34                 set("NomeMae", soapEnvelope.substring(inicio,fim));
35             }
36
37             inicio = soapEnvelope.indexOf("NomePai");
38             if(inicio != -1){
39                 inicio = soapEnvelope.indexOf("=", inicio);
40                 inicio += 1;
41                 fim = soapEnvelope.indexOf(";", inicio);
42                 set("NomePai", soapEnvelope.substring(inicio,fim));
43             }
44             ...
45         }
46     }
47         ...
48 }
```

Listagem 38: Código da classe CondutorInformacoesClasse.

O código da Listagem 38 apresenta a recuperação de dados através da busca por informações na variável `soapEnvelope` sendo parâmetro do método construtor da classe, conforme linha 3. Esta Listagem tem o mesmo princípio e lógica da Listagem 27, alterando apenas os dados a serem recuperados.

As linhas 5 a 11 são responsáveis por recuperarem o valor `IdPessoa`, sendo este um número sequencial único designado a identificar o condutor na base de dados do Detran-TO. O método `set()` é responsável por armazenar este dado através da chave `“IdPessoa”`.

A recuperação do CPF é verificada nas linhas 13 a 19 através da busca pela palavra CPF. O método `set()` armazena este dado através da chave `“CPF”`. O nome do condutor é recuperado através das linhas 21 a 27, sendo armazenado através do método `set()` pela chave `“Nome”`. O nome da mãe é adquiro através das linhas 29 a 35 pelo parâmetro `“NomeMae”` e armazenado através do método `set()` pela chave `“NomeMae”`. O nome do pai é recuperado através das linhas 37 a 43 pela busca da palavra `“NomePai”` na variável `soapEnvelope` e armazenado utilizando o método `set()` tendo a chave de parâmetro a palavra `“NomePai”`. A busca é realizada para cada chave definida na Listagem 37, recuperando todos os dados do condutor contidos na base de dados do Detran-TO.

O método `set()` da classe `CondutorInformacoesClasse` é apresentado na Listagem 39.

```
1  public class CondutorInformacoesClasse {
2
3      public void set(Object k, Object v){
4
5          String key = (String) k;
6          String value = (String) v;
7
8          if (KEY_CPF.equals(key))
9              CPF = value.trim();
10         else if (KEY_IdPessoa.equals(key))
11             IdPessoa = Integer.parseInt(value.trim());
12         else if (KEY_DescricaoRetorno.equals(key))
13             DescricaoRetorno = value.trim();
14         else if (KEY_Nome.equals(key))
15             Nome = value.trim();
16         else if (KEY_NomeMae.equals(key))
17             NomeMae = value.trim();
18         else if (KEY_NomePai.equals(key))
19             NomePai = value.trim();
20
21         ...
22     }
23 }
```

Listagem 39: Método SET da classe CondutorInformacoesClasse.

Os parâmetros recebidos pelo método `set()` são a chave, `Object k`, e o valor, `Object v`, conforme visto na linha 3 da Listagem 39. As variáveis `key` e `value` são responsáveis por armazenarem os valores `k` e `v`, respectivamente, transformando-os em `String`.

As estruturas `if else if` apresentadas nas linhas 8 a 19 verificam a igualdade das chaves com as variáveis declaradas no método. Caso tenham sucesso na verificação, a informação é armazenada na variável correspondente ao dado. Um exemplo a ser apresentado é a verificação da chave `KEY_CPF` com o valor da variável `key`. Esta verificação sendo verdadeira significa que os dados existentes são referentes ao CPF do condutor, resultando no armazenamento deste dado contido na variável `value` na variável de classe CPF. Este procedimento é repetido para todos os dados existentes de condutor.

O método de recuperação de dados, `get()`, é apresentado na Listagem 40.

```
1  public class CondutorInformacoesClasse {
2
3      public String get(Object k){
4
5          String key = (String) k;
6
7          if (KEY_CPF.equals(key))
8              return CPF.toString();
9          else if (KEY_IdPessoa.equals(key))
10             return String.valueOf(IdPessoa);
11         else if (KEY_DescricaoRetorno.equals(key))
12             return DescricaoRetorno.toString();
13         else if (KEY_Nome.equals(key))
14             return Nome.toString();
15         else if (KEY_NomeMae.equals(key))
16             return NomeMae.toString();
17         else if (KEY_NomePai.equals(key))
18             return NomePai.toString();
19
20         return null;
21
22         ...
23     }
24 }
```

Listagem 40: Método GET da classe CondutorInformacoesClasse.

A Listagem 40 finaliza a classe `CondutorInformacoesClasse` com a apresentação do método `get()`. Este método recebe o parâmetro `Object k`, representante da informação a ser recuperada. A linha 5 armazena este valor na variável `key`. As estruturas `if else if`, apresentadas nas linhas 7 a 20, verificam o sucesso da comparação entre as chaves declaradas na classe e a variável `key`. Havendo sucesso nesta checagem, o dado solicitado é retornado. Não sendo encontrada nenhuma chave correspondente, é retornado o valor `null`.

4.6. Método e Classes Globais

Os métodos e classes apresentados nesta seção são os códigos que estão presentes em vários momentos da descrição dos serviços apresentados no decorrer da Seção 4. São eles:

- método `internetConnection()`: responsável por verificar a conexão da aplicação com a Internet;

- classe `VeiculoInfo`: classe responsável por conter a estrutura dos dados referentes a veículos que são apresentados ao usuário;
- classe `CondutorInfo`: classe responsável por conter a estrutura dos dados referentes a condutores que são apresentados ao usuário.

4.6.1. Especificação do método internetConnection()

O método `internetConnection()` verifica a conexão da aplicação com a Internet e retorna a informação ao contexto solicitante. A Listagem 41 apresenta o código deste método.

```
1  public static boolean internetConnection(Context context) {
2      ConnectivityManager connMgr =
3          (ConnectivityManager)context.getSystemService(Context.CONNECTIVITY_SERVICE);
4      NetworkInfo info=connMgr.getActiveNetworkInfo();
5      return(info!=null && info.isConnected());
6  }
```

Listagem 41: Método internetConnection().

O método representado na Listagem 41 recebe como parâmetro o contexto atual da aplicação, sendo este representado pela variável `context` do tipo `Context`. A classe `ConnectivityManager` é responsável por conter todos os métodos necessários para o manuseio da conectividade da aplicação. Através dessa classe é possível verificar a conexão da aplicação com a Internet.

A linha 2 é responsável pela criação da variável `connMgr`, sendo a linha 3 responsável por adquirir o serviço referente a conectividade. A linha 4 cria a variável `info` do tipo `NetworkInfo` recebendo o estado atual da conexão através do método `getActiveNetworkInfo()` existente na variável `connMgr`. O dado referente à existência de conectividade com a Internet é retornado ao solicitante, conforme linha 5.

A próxima seção apresenta a classe VeiculoInfo, que consiste na definição dos dados responsável por armazenar os

4.6.2. Especificação da classe VeiculoInfo

A classe `VeiculoInfo` é responsável por armazenar os valores apresentados em cada célula da lista de dados. A Listagem 42 apresenta o código desta classe.

```
1  public class VeiculoInfo {
2
3      private String titulo;
4      private String resultado;
5      private int opcao;
6
7      public String getTitulo() {
8          return titulo;
9      }
10     public void setTitulo(String titulo) {
11         this.titulo = titulo;
12     }
13
14     public String getResultado() {
15         return resultado;
16     }
17     public void setResultado(String resultado) {
18         this.resultado = resultado;
19     }
20
21     public int getOpcao() {
22         return opcao;
23     }
24     public void setOpcao(int opcao) {
25         this.opcao = opcao;
26     }
27 }
```

Listagem 42: Classe VeiculoInfo.

A classe `VeiculoInfo`, apresentada na Listagem 42, apresenta 3 valores básicos, sendo eles o título da informação, o resultado e a opção, conforme linhas 3 a 5. Esta classe é necessária para armazenar os valores recuperados dos servidores do Detran-TO. Cada informação recuperada é do tipo `VeiculoInfo`, ou seja, cada elemento é um objeto desta classe. Assim, após recuperar todas as informações e as colocar em uma lista do tipo `VeiculoInfo`, é possível a apresentação dos dados ao usuário.

A classe possui os métodos `get()`, responsáveis por recuperar o dado desejado, e os métodos `set()`, responsáveis por definir um valor às variáveis de classe. Esses valores são apresentados ao usuário como sendo as informações existentes na base de dados do Detran-TO.

A próxima seção apresenta a classe `CondutorInfo`, responsável por armazenar as informações relacionadas ao condutor consultado.

4.6.3. Especificação da classe CondutorInfo

A classe `CondutorInfo` a responsável por armazenar os valores apresentados em cada célula da lista de dados. A Listagem 43 apresenta o código desta classe.

```
1  public class CondutorInfo {
2
3      private String titulo;
4      private String resultado;
5      private int opcao;
6
7      public String getTitulo() {
8          return titulo;
9      }
10     public void setTitulo(String titulo) {
11         this.titulo = titulo;
12     }
13
14     public String getResultado() {
15         return resultado;
16     }
17     public void setResultado(String resultado) {
18         this.resultado = resultado;
19     }
20
21     public int getOpcao() {
22         return opcao;
23     }
24     public void setOpcao(int opcao) {
25         this.opcao = opcao;
26     }
27 }
```

Listagem 43: Classe CondutorInfo.

A classe `CondutorInfo` apresentada na Listagem 43 se refere às informações recuperadas do condutor consultado. Esta classe possui as mesmas atribuições da classe `VeiculoInfo`, apresentada na Listagem 42. Optou-se por separar as atribuições visto que, futuramente, diferentes estruturas referentes às informações poderiam surgir, diferenciando veículos de condutores.

5. CONSIDERAÇÕES FINAIS

Este trabalho apresentou o desenvolvimento de uma aplicação para recuperação de dados de veículos, a nível nacional, e condutores, a nível estadual, nas blitz realizadas no Estado do Tocantins. Esta iniciativa visa amenizar os problemas existentes na aferição da documentação dos condutores, visto que a apresentação dos dados atualmente se restringe a documentos impressos, tornando o método propício a falhas. Desta forma, há, naturalmente, a possibilidade de falsificação de documentos como, por exemplo, CNH e/ou CRLV.

Através da aplicação desenvolvida, a Polícia Militar tem um suporte em tempo real com informações verdadeiras e em tempo hábil para auxílio em suas iniciativas. O acesso aos dados se torna simples e prático, adquiridos em poucos passos.

A arquitetura do aplicativo foi desenvolvida de forma que permita a adição de qualquer função existente atualmente no Detran-TO e/ou até mesmo elaborar novas funcionalidades que sejam necessárias, de acordo com eventuais necessidades que possam surgir. Isto se deve principalmente pela utilização da tecnologia de *Web Services*, que torna possível a comunicação com sistemas externos, independente de plataforma de *hardware* e *software*, como Sistema Operacional, plataforma de desenvolvimento, dentre outros.

Funcionalmente, o aplicativo desenvolvido teve sua estrutura baseada em consulta por informações referentes aos veículos que envolveu todo o território Nacional, mas com uma ênfase maior nas informações relacionadas aos condutores do Estado do Tocantins. Esta ênfase ocorreu pelo fato do desenvolvimento do aplicativo ter sido focado na utilização por parte da Policia Militar do Estado. Estas pesquisas foram codificadas por meio da utilização da linguagem de programação Java para Android. A partir do aplicativo desenvolvido e do *Web Service* implementado, as informações que o aplicativo disponibiliza foram extraídas diretamente da base de dados do Detran-TO e apresentadas de forma imediata na tela do aplicativo.

Com relação às tecnologias utilizadas no desenvolvimento deste trabalho, foi utilizada a plataforma de desenvolvimento para dispositivos móveis Android a fim de possibilitar o acesso às informações independentes da localidade que os Policiais se encontram ao realizarem uma fiscalização. Consequentemente, pela proposta do trabalho ter se baseado no

desenvolvimento para Android, utilizou-se a linguagem de programação Java para codificar os métodos de consulta e apresentação de informações. O *Web Service* implementado foi utilizado como canal de extração de informações da base de dados do Detran-TO para serem exibidas no aplicativo desenvolvido.

A partir do exposto até o momento, pode-se dizer que a aplicação deste trabalho apresenta todos os dados necessários ao usuário de forma simples, prática e de fácil entendimento, tendo em vista que as dimensões dos dispositivos móveis em geral são reduzidas. As informações apresentadas na tela do aplicativo se mostraram fundamentais e suficientes para os Policiais que utilizam o aplicativo desenvolvido, abrangendo todos os dados existentes sobre um veículo e/ou condutor.

A utilização do aplicativo desenvolvido e apresentado neste trabalho está em fase experimental, sendo utilizado pela Polícia Militar em blitz para testes e críticas. As informações disponibilizadas pelo aplicativo são mais amplas se comparadas às que os Policiais possuíam, obtidas apenas ao conferir os documentos exigidos pela Lei Brasileira de Trânsito, a CNH e o CRLV. A partir da utilização do aplicativo desenvolvido, os Policiais possuem informações adicionais às que constam nos documentos, como a existência de multas e apreensão sobre o veiculo, bem como a ocorrência de furto e taxas não pagas junto ao Dentran-TO, dentre outras informações relacionadas à especificação detalhada do veículo como cor, modelo, ano, quantidade de lugares, dentre outras.

Outro fato que pode ser levado em consideração ao afirmar que o aplicativo desenvolvido é um caso de sucesso, foi o interesse por parte da ATTM (Agência de Trânsito, Transporte e Mobilidade) em utilizar o aplicativo, uma vez que esta Agência também tem poder regulatório em vias públicas no Estado do Tocantins. A ATTM, ao acompanhar a Polícia Militar durante as blitz, notou a eficiência e a riqueza de detalhes apresentadas na aplicação, indispensáveis na fiscalização eficaz de condutores e veículos automotores.

Ao analisar as contribuições que o aplicativo desenvolvido proporciona a seus usuários, no caso a Polícia Militar do Estado do Tocantins, pode-se enumerar:

- mobilidade, de forma que a qualquer momento e lugar os Policiais podem obter informações remotas que desejarem;
- interface gráfica simples de se manusear, quando comparado à computadores, apesar de não possuírem a mesma flexibilidade;
- facilidade de uso, de forma que por meio de um conjunto mínimo de interações a Polícia Militar tenha acesso às informações úteis e suficientes para uma melhor fiscalização de veículos e condutores;

- facilidade em transporte, que se deve à pequena dimensão que os aparelhos móveis possuem, com a possibilidade de manter o sistema conectado;
- acesso às informações atualizadas e disponíveis em tempo real, relacionadas a legitimidade de documentos, pendências existentes no Detran-TO, ocorrência de roubo/furto e apreensão do veiculo, dentre outras informações.

O aplicativo apresentado neste trabalho teve como foco o atendimento às necessidades do Estado do Tocantins. Entretanto, nada impossibilita que o aplicativo seja utilizado nos demais Estados do País da forma como está disponível hoje, uma vez que disponibiliza a consulta por informações a nível Nacional. Como o aplicativo atualmente possui em sua busca as opções de consulta por nível Estadual e Nacional, no caso deste ser disponibilizado em âmbito Nacional, seria interessante apenas retirar as opções de busca estadual e fazer com que o aplicativo sempre extraia informações da Base Nacional ou modificar a busca estadual de tal forma que os dados sejam recuperados por estado. Vale ressaltar que a não adaptação do aplicativo não impede seu funcionamento em outros Estados, mas seria uma adaptação para melhoria de usabilidade, visto que as opções de busca estadual não seriam utilizadas.

Como proposta para trabalhos futuros, pretende-se desenvolver módulos adicionais para o aplicativo e, assim, naturalmente, atender gradativamente às necessidades dos Órgãos fiscalizadores de trânsito. Um exemplo de módulo a ser adicionado no aplicativo se refere ao gerenciamento de multas. Dentro deste módulo, pretende-se disponibilizar as seguintes funcionalidades:

- aplicação de multas: esta funcionalidade objetiva possibilitar que a Polícia Militar ao realizar uma blitz, possa aplicar uma multa ao condutor, quando necessário, por meio do aplicativo Android. Sendo assim, a partir do momento que uma multa é aplicada, esta deverá ser armazenada diretamente na base de dados do Detran-TO, ficando disponível desta forma também para os sistemas internos do Órgão;
- impressão de multas: esta funcionalidade visa disponibilizar à Policia Militar a opção de imprimir um boleto referente à multa aplicada durante a realização de uma fiscalização de forma que esta seja entrega imediatamente ao condutor. Pretende-se que esta impressão seja realizada através de uma impressora *Bluetooth* acionada diretamente pela aplicação.

REFERÊNCIAS BIBLIOGRÁFICAS

ALBINADER, J. A; LINS, R. D. **Web Services em Java.** Rio de Janeiro: Brasport, 2006. Disponível em: <http://books.google.com.br/books?hl=pt-BR&lr=lang_pt&id=MUdQSavIynEC&oi=fnd&pg=PA1&dq=web+services&ots=SAq3XlUYIu&sig=gFwQRL_omif4jZ_cSULrMLlvbJM#v=onepage&q=web%20services&f=false>. Acessado em: março 2012.

AMORIM, A. D. **ANDROID, uma visão geral.** 2011. Disponível em: <http://pt.scribd.com/doc/60326457/ANDROID-uma-visao-geral-1-0>. Acessado em: abril 2012.

ANDROID SDK. Disponível em: <http://code.google.com/android/>. Acessado em: março 2012.

BASIURA, R. et. al. **Professional ASP.NET Web Service.** São Paulo: Pearson Education, 2003.

BECKER, A. K., CLARO, D. B., SOBRAL, J. B. **Web Services e XML:** Um Novo Paradigma da Computação Distribuída. 2002. Monografia (Graduação em Ciências da Computação) – Universidade Federal de Santa Catarina, Florianópolis. Disponível em: <http://homes.dcc.ufba.br/~dclaro/download/ArtigoWebServices.pdf>. Acessado em: março 2012.

CBDI Web Services Roadmap, **The Web Services Protocol Stack.** 2005. Disponível em: <http://roadmap.cbdiforum.com/reports/protocols>. Acessado em: março 2012.

DAUM, B. **Arquitetura de sistemas com XML:** conteúdo, processo e implementação. Rio de Janeiro: Campus, 2002. Disponível em: <http://www.argonavis.com.br>. Acessado em: março 2012.

FILHO, O. V. S.; ZARA, P. M. **Microsoft .NET: uma visão geral para programadores.** São Paulo: SENAC São Paulo. 2002.

HENDRICKS, M. et. al. Profissional Java Web Service. Rio de Janeiro: Editora Alta Books, 2002.

JEWELL, T; CHAPPELL, D. **UDDI:** Universal Description, Discovery, and Integration, Part 1. Ther O'Reilly Network. Disponível em: <http://www.oreillynet.com/pub/a/onjava/excerpt/jws_6/index1.html>. Acessado em: março 2012.

LEOPOLDO, M. R. B. **Simple Object Access Protocol:** Entendendo o Simple Object Access Protocol (SOAP). Disponível em: <http://wiki.pge.ce.gov.br/images/0/0b/SOAP.pdf>. Acessado em: março 2012.

LIMA, B. M.; LIMA, D. B.; OLIVEIRA, E. S. **Redução do Acoplamento com Frameworks Específicos de Plataforma no Mdarte:** Estudo de Caso em Ambientes Móveis. 2011. Monografia (Graduação em Ciência da Computação) – Universidade Federal do Rio de Janeiro, Rio de Janeiro. Disponível em: <http://www.cos.ufrj.br/~diogobor/files/projetoFinal.pdf>. Acessado em: março de 2012.

MACK, R. S. **Sistema de recomendação baseado na localização e perfil utilizando a plataforma android.** 2010. Monografia (Graduação em Ciência da Computação) – Universidade Federal do Rio Grande do Sul, Porto Alegre. Disponível em: <http://www.lume.ufrgs.br/bitstream/handle/10183/28328/000767836.pdf?sequence=1>. Acessado em: abril de 2012.

MARTINS, R. J. W. A. **Desenvolvimento de Aplicativo para Smartphone com a Plataforma Android.** 2009. Monografia (Graduação em Engenharia de Computação) – Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro. Disponível em: <http://www.icad.puc-rio.br/~projetos/android/files/monografia.pdf>. Acessado em: abril 2012.

OHA. **Alliance Overview.** Disponível em: <http://www.openhandsetalliance.com/oha_overview.html>. Acessado em: março 2012a.

OHA. **Android Overview.** Disponível em: <http://www.openhandsetalliance.com/android_overview.html>. Acessado em: março 2012b.

PEREIRA, L. C. O.; SILVA, M. L. da. **Android para desenvolvedores.** Rio de Janeiro: Brasport, 2009. Disponível em: <http://books.google.com.br/books?hl=pt-BR&lr=&id=8u9wJowXfdUC&oi=fnd&pg=PA1&dq=plataforma+android&ots=LSfl_3Yom4&sig=QsRzHqf2rIwITvKa450PbDCPlBg#v=onepage&q&f=false>. Acessado em: março 2012.

PROJECT, A. O. S. **Android –** an open handset alliance project, 2009. Disponível em: <http://developer.android.com/sdk/android-1.5.html>. Acessado em: março, 2012.

ROSENBERG, J.; REMY, D. L. **Securing Web Services with WS-Security.** 1. Ed. Indianopolis: Sams, 2004. 408 p. *apud* SILVA R. F.; CUNHA, J. A. **Arquitetura de Segurança em Aplicações Baseadas em Web Service**. In: Holos, Ano 21, dezembro 2005, 15. Disponível em: <http://www2.ifrn.edu.br/ojs/index.php/HOLOS/article/view/77/82>. Acessado em: março 2012.

SAMPAIO, C. **SOA e Web Services em Java.** São Paulo: Brasport. 2006.

SIMÃO, A. M. de L. et. al. **Aquisição de Evidências Digitais em *Smartphones* Android.** In: Procedings of the Sixth International Conference on Forensic Computer Science Investigation ICoFCS 2011 (2011). Disponivel em: <www.icofcs.org/2011/ICoFCS2011-PP09.pdf>. Acessado em: março 2012.

UDDI, **Programmer's API 1.0** UDDI.org, jun 2002. Disponível em: <http://uddi.org/pubs/ProgramamersAPI-V1.01-Published-20020628.pdf>. Acessado em: março 2012.

W3C. **Web Services Description Language (WSDL) 1.1.** 2001. Disponível em: <http://www.w3.org/TR/wsdl>. Acessado em: março 2012.

W3C. **Simple Object Access Protocol (SOAP) 1.1.** 2000. Disponível em: <http://www.immagic.com/eLibrary/ARCHIVES/SUPRSDED/W3C/W000520N.pdf>. Acessado em: março 2012.

XML, **Extensible Markup Language (XML).** Word Wide Web Consortion. Disponível em: <http:www.w3.org/XML/>. Acessado em: março 2012.

APÊNDICE A – Procedimento de configuração do Ambiente para o Desenvolvimento de Aplicativos Android

Esta seção apresenta como baixar e instalar o Android SDK e o *plugin* ADT, bem como realizar a configuração do ambiente de desenvolvimento a fim de possibilitar o desenvolvimento de aplicativos Android.

Abaixo segue a lista de todos os recursos, ferramentas, *plugins*, entre outras tecnologias que foram utilizadas no desenvolvimento do aplicativo proposto neste trabalho:

- Windows 7;
- Eclipse Java EE IDE for Web Developers;
- *Plugin Android Development Tools*.

Instalações

Esta seção será responsável pelo auxílio à configuração e instalação da IDE Eclipse com o Android SDK, sendo este o *software* que será utilizado para criação da aplicação referida neste trabalho.

IDE Eclipse

Para a instalação da IDE Eclipse, o arquivo referente ao Eclipse pode ser encontrado no site do fabricante, na seção de downloads, http://www.eclipse.org/downloads/index.php.

A IDE Eclipse não é instalável, ou seja, não existe um passo a passo de instalação como a maioria das aplicações do Windows. Após ser realizado o download do arquivo, encontrado no site do fabricante, é necessária a extração dos dados para uma pasta destino e a execução do Eclipse é realizada através do arquivo Eclipse.exe. Seguindo estes passos, a IDE está apta para ser utilizada.

Ao iniciar o Eclipse, será necessário informar a pasta que armazenará o *workspace*, como visto na Figura . Este workspace é responsável por guardar todos os projetos construídos com o auxílio da IDE Eclipse.

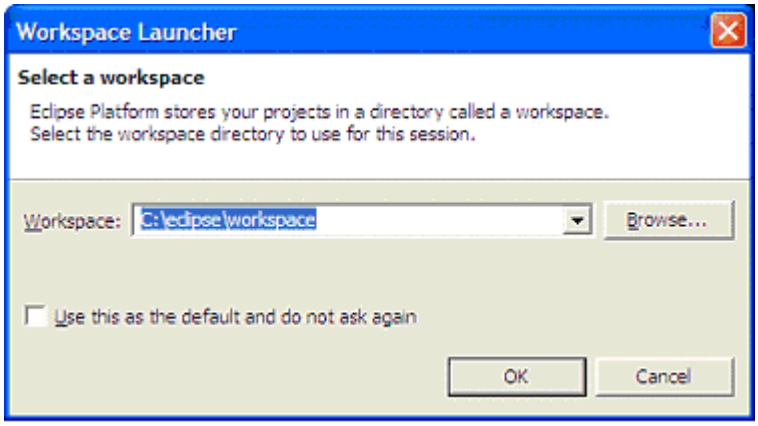

Figura 1: Definindo o workspace da IDE Eclipse.

Após a definição do workspace é necessário clicar no botão OK para a inicialização da IDE. Feito estes passos, a IDE está pronta para utilização, conforme Figura.

Figura 2: Tela inicial da IDE Eclipse.

Android SDK

Para a instalação do Android SDK, será necessário o acesso ao site http://developer.android.com/sdk/index.html e efetuar o download do arquivo correspondente.

Após o download do Android SDK, é necessária a descompactação do arquivo, gerando assim uma pasta contendo os arquivos necessários. Esta pasta gerada contém vários subdiretórios, sendo elas *tools*, *samples*, entre outras.

Realizada esta etapa, é necessário configurar o Android SDK na IDE Eclipse. Os passos realizados para a configuração são:

1. Selecionar a opção Help > Software Updates > Find and Install;
2. Na janela que surgir é necessário selecionar a opção “Search for new features to install” através do botão Next;
3. Selecione “New Remote Site”;
4. Entre com um nome de identificação e informe o site “https://dl-ssl.google.com/android/eclipse/” e clique em OK;
5. Através deste repositório, serão exibidas ao usuário todas as instâncias para download. Caso necessite a instalação de todos, escolha Install All;
6. Reinicie a IDE Eclipse;
7. Após a reinicialização é necessário a atualização das preferências para que o Eclipse identifique a pasta onde foi descompactado os arquivos do Adrnoid SDK;
8. Processo finalizado.