Desenvolvimento de um Web Service para Gerenciamento de Frotas de Taxis

Tales Luiz Bortolin[1] , Ricardo Frohlich da Silva[1]

[1]Sistemas de Informação - Centro Universitário Franciscano (UNIFRA)
97.010.032 - Santa Maria - RS - Brasil

talesbortolin@unifra.edu.br, ricardo.frohlich@unifra.br

Abstract. *This article presents the development of a Web Service that provides developers a service to manage locations of taxi fleets through a call control in commercial environments or by providing validation services in an Android application.*

Resumo. *Neste artigo é apresentado o desenvolvimento de um Web Service que disponibiliza aos desenvolvedores, um serviço para gerenciar a localização de frotas de taxis por meio do controle de chamadas em ambientes de comércio ou prestação de serviço com validação em uma aplicação Android.*

1. Introdução

Atualmente os consumidores estão cada vez mais exigentes e devido ao acúmulo de funções e atividades, estão acostumados a realizar suas tarefas no conforto de suas casas, em minutos, apenas utilizando um *tablet*, *smartphone* ou um *notebook* conectado à internet, sem a necessidade de enfrentar o trânsito congestionado e filas enormes em bancos e lojas, por exemplo [Fincotto e Santos 2014].

A utilização unificada de processos encontrados em diferentes sistemas e escritos em diferentes linguagens, são alguns exemplos de carências encontradas nos dias de hoje, quando se fala em integração entre aplicações. A fim de sanar estas questões, a tecnologia dos *Web Services* foi criada, permitindo assim, disponibilizar formas de integrar sistemas distintos, modularizar serviços e capacitar a integração e consumo de informações [Moro, Dorneles e Rebonatto 2011].

Os *Web Services* são serviços disponibilizados na internet considerados um dos principais meios de comunicação entre sistemas de diferentes plataformas de *hardware* e *software*. De um modo geral, a comunicação entre as aplicações para dispositivos móveis como: *Android*, *iOS* e *Windows* e os servidores, é realizada com a utilização desses serviços [Dantas 2007].

Por este motivo, este estudo tem como objetivo, o desenvolvimento de um *Web Service* (WS) para gerenciamento de frotas de taxis, o que torna o processo de gerenciamento mais prático e ágil. Na seção seguinte, observa-se os objetivos que regem esse estudo.

1.1. Objetivo Geral

O presente trabalho tem como objetivo principal, a implementação de um *Web Service*, o qual consiste no gerenciamento de frotas de taxis, por meio do controle de chamadas, utilizando o trabalho desenvolvido por Franco (2013), para a sua validação.

1.2. Objetivos Específicos

O presente trabalho tem os seguintes objetivos específicos:

- Desenvolver um *Web Service* baseado no protocolo SOAP;
- Publicar o serviço utilizando o servidor IIS;
- Armazenar os dados das aplicações em um banco de dados;
- Adaptar o *Web Service* para trabalhar com a aplicação de Franco (2013).
- Validar a proposta do trabalho através de um estudo de caso.

2. Referencial Teórico

Nesta seção, serão tratados dos referenciais bibliográficos que envolvem este trabalho. Dentre eles, é possível destacar alguns que são mais relevantes como: *Web Service*, Geolocalização e o Android.

2.1. Web Service

Segundo Reverbel (2006), Web Service é: "*nome dado à tecnologia que permite a comunicação entre aplicações de uma maneira independente de sistema operacional e de linguagem de programação*".

De acordo com Lopes e Ramalho (2004), o ciclo de vida de um *Web Service* compreende quatro estados distintos, sendo eles [Lopes e Ramalho 2004]:

- Publicação: Processo, opcional, através do qual o fornecedor do *Web Service* disponibiliza seu serviço, efetuando o registo do mesmo no repositório de Web Services (UDDI).
- Descoberta: Processo, opcional, através do qual uma aplicação cliente toma conhecimento da existência do *Web Service* pretendido, pesquisando num repositório UDDI.
- Descrição: Processo pelo qual o *Web Service* expõe a sua API (documento WSDL); desta maneira a aplicação cliente tem acesso a toda a interface do *Web Service*, onde se encontram descritas todas as funcionalidades por ele disponibilizadas, assim como os tipos de mensagens que permitem aceder às ditas funcionalidades.
- Invocação: Processo pelo qual cliente e servidor interagem, através do envio de mensagens de *input* e de eventual recepção de mensagem de *output*.

2.1.1. Arquitetura SOAP

O SOAP (*Simple Object Access Protocol*) é um padrão onde as requisições para o *Web Service* são feitos no formato XML (*Extensible Markup Language*) e permitem recursos

mais sofisticados como passagem de estruturas e *arrays*. Independente de como seja feito o pedido, as respostas são sempre em XML. O XML descreve os dados em tempo de execução e evita problemas causados por mudanças imprevistas nas funções, já que os objetos chamados têm a possibilidade de sempre validar os argumentos das funções, tornando o protocolo robusto [Sant'anna 2015].

A arquitetura SOAP descreve quatro componentes principais: a formatação de convenções para encapsular dados e orientações de rotas na forma de um envelope, um transporte ou protocolo, regras de codificação e um RPC (*Remote Procedure Call*) [Ferreira e Mota 2014].

Na Figura 1, pode-se observar uma breve amostra de como é o funcionamento do *Web Service* baseado na arquitetura SOAP.

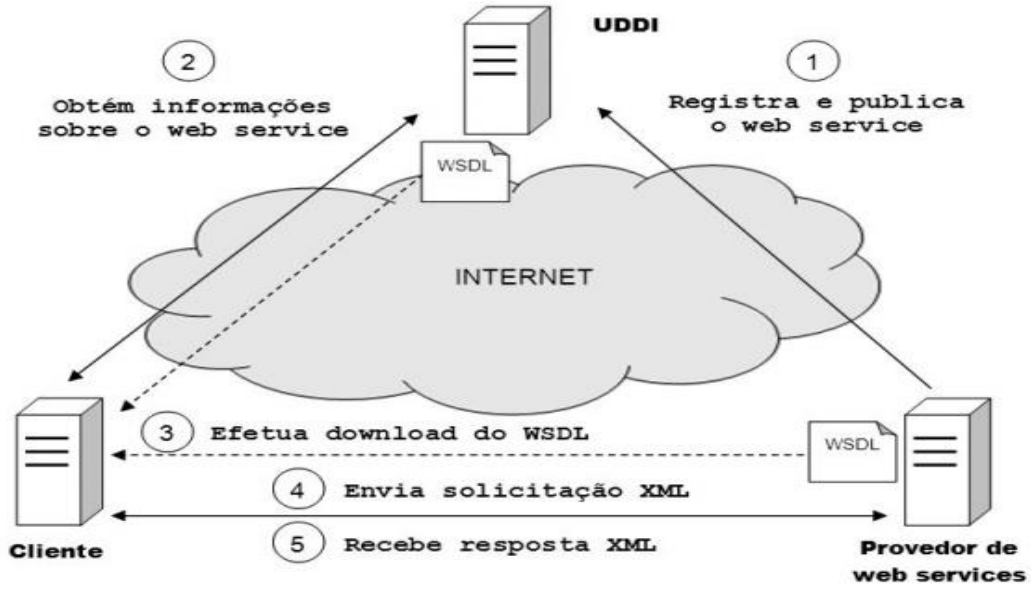

Figura 1. Arquitetura Web Service SOAP [Gomes 2014].

Descrevendo cada item numerado na Figura 1, pode-se ter um entendimento mais aprofundado da sequência das operações [Gomes 2014]:

1. Registra e publica o *Web Service*: os *Web Services* ficam armazenados junto de seus descritores (WSDL) em provedores de *Web Services.* Após a criação de um *Web Service,* é necessário registrá-lo e publicá-lo em algum diretório de registro de *Web Services*, também conhecido como UDDI (*Universal Description, Discovery and Integration*).
2. Obtém informações sobre o *Web Service*: quando um desenvolvedor de software necessita consumir o serviço de um *Web Service*, ele efetuará uma pesquisa pelo tipo de *Web Service* que precisa. O UDDI fornece para o desenvolvedor, o endereço do *Web Service* e o arquivo WSDL.
3. Efetua *download* do WSDL: assim que o desenvolvedor obtiver as URIs do *Web Service* e de seu descritor, já é possível efetuar o *download* do arquivo WSDL e, em seguida, efetivamente consumir o *Web Service*, onde o *software* cliente fará uma solicitação e obterá uma resposta conforme o serviço solicitado.

4. Enviar solicitação XML: o *software* cliente fará chamadas ao *Web Service* por meio de seu URI, enviando solicitações no formato XML.
5. Receber resposta XML: O *Web Service* receberá a solicitação anterior, realizará um determinado tipo de processamento e retornará um resultado ao *software* solicitante no formato XML.

O WSDL (*Web Service Description Language*) é um arquivo XML que informa para o cliente, como utilizar o serviço, ou seja, nele contém o nome dos métodos, nome de parâmetros, endereço do serviço, enfim, todo o conteúdo necessário para utilizar o mesmo [Ferreira e Mota 2014].

O UDDI (*Universal Description, Discovery and Integration*) é um mecanismo que atende tanto ao cliente quanto ao provedor (Servidor). Ele fornece ao provedor de *Web Services,* meios para que estes sejam registrados e publicados, permitindo assim que eles sejam pesquisados e encontrados pelos clientes [Gomes 2014].

O SDK do *Android* não possui uma solução embutida para consumir *Web Services*, portanto, é necessário incorporar a biblioteca KSOAP2 que é um *Framework* para acesso à *Web Services,* sem a necessidade de geração de códigos ou uso de *proxies* dinâmicos [Ushisima 2011].

Por se tratar de uma biblioteca para dispositivos móveis, ela possui um nível computacional "leve" e de pequena abrangência, mas que desempenha muito bem sua função de interoperar com *Web Services*, possibilitando a passagem de parâmetros complexos [Medeiros 2014].

2.2. Android

Segundo Lecheta (2010) o *Android* é um sistema operacional baseado no *kernel 2.6* do Linux [Linux 2006] que é responsável por gerenciar a memória, os processos, threads, segurança dos arquivos, além das redes e *drivers*. Sendo assim, sua plataforma comporta o desenvolvimento de aplicações customizadas que podem ser executadas em diferentes tipos de dispositivos. Uma vantagem é a possibilidade de poder desenvolver usando uma plataforma que possui seu código aberto (*Open-Source*), o que indiretamente, acaba contribuindo para o crescimento do *Android*.

O Google lançou recentemente a sua nova IDE (*Integrated Development Environment*) de programação para *Android*, o Android Studio [Android Studio 2014], totalmente integrada e remodelada, não sendo mais necessário incorporar as ferramentas do SDK (*Software Development Kit*) e o *plugin* do AVD (*Android Virtual Device*) para simular a interface do dispositivo móvel. Essas ferramentas já vêm incorporadas na própria IDE com funções de edição inteligente de código, recursos para *design* de interface de usuário, análise de performance, entre outros [Rocha 2014].

2.3. Geolocalização

A Geolocalização é uma tecnologia que utiliza os dados do dispositivo móvel para indicar a localização no globo terrestre, da mesma forma que um aparelho GPS [Ferreira 2014].

Com o grande crescimento no ramo empresarial, a tecnologia da geolocalização trouxe muitos benefícios como, por exemplo, as empresas de transporte que conseguem monitorar sua frota 24 horas por dia com o objetivo de proteger suas cargas; as empresas de táxis podem controlar suas frotas onde com um aplicativo instalado no celular, consegue-se mapear todos os veículos. Para os usuários comuns, podem encontrar postos de combustíveis mais próximos usando um aplicativo específico no celular, entre diversos outros exemplos [Ferreira 2014].

2.4. Trabalhos Relacionados

Nesta seção, serão tratados assuntos que possuem relação com o presente trabalho.

2.4.1. Comparando Aplicação Web Service REST e SOAP

A proposta do trabalho de Ferreira e Mota (2014) trata sobre o que é um serviço e as principais características das arquiteturas REST e SOAP.

Segundo eles, serviços são considerados como blocos de construção independentes que, coletivamente formam um ambiente de aplicação. E uma das principais características é a completa autonomia em relação aos outros serviços, ou seja, cada serviço é responsável pelo seu próprio domínio, que limita seu alcance a uma função de negócio específico.

A arquitetura SOAP tem como principais características: estabelecer um formato padrão de mensagens que consistem em documento XML, capaz de hospedar dados RPC, facilitando o intercâmbio de dados de modelos síncrono e assíncronos; descrever funções e tipos de dados; suportar o protocolo SOAP, de forma nativa, várias linguagens; formatar em XML, grande quantidade de conteúdo textual.

Já, na arquitetura REST, não existe um descritor de funcionamento do serviço. A requisição realizada pelo cliente, parte do princípio de que a mesma conhece o que deve ser enviado para o servidor, facilitando assim, o processo de implementação. São algumas características da arquitetura REST: apresentar pequenas quantidades de conteúdo textual formatado em XML ou JSON; na maioria dos casos, operar sobre o protocolo HTTP; ser indicada para uso onde há limitação da largura de banda.

2.4.2. Mobilidade no Monitoramento de Subestações Elétricas através de Serviços Web

No trabalho de Padoin (2007), foi desenvolvido um sistema de automação de subestações de energia elétrica que faz uso de unidades remotas de aquisição de dados onde a comunicação destas unidades pode ser realizada via rádio (*wifi*), RS323 (Protocolo de Comunicação Serial), RS458 (Protocolo de Comunicação Serial) ou através de redes do tipo *ethernet*. Com o intuito de expandir o sistema e possibilitar que o mesmo possa ser acessado através de equipamentos móveis, foi desenvolvido um *Web Service*.

Este *Web Service* foi desenvolvido baseado na arquitetura SOAP e utilizado também o pacote JWSPD, que é um pacote específico para a criação de *Web Services* com *Java* no modelo de comunicação RPC que fornece todo o suporte de comunicação entre as aplicações cliente e servidor.

Dispondo destes recursos, as funções e a operação da subestação podem ser monitoradas e controladas de qualquer ponto remoto através de equipamentos móveis que tenham acesso à internet.

2.4.3. Sistema para Controle de Horários em Quadras Esportivas

O trabalho de Isernhagen (2014) trata do desenvolvimento de um sistema para controle de horários em quadras de esportes.

Neste trabalho foram desenvolvidos dois sistemas: um sistema para dispositivo móvel, utilizando a linguagem de programação Java, e outro sistema *web*, utilizando as linguagens de programação PHP, HTML, CSS e JAVASCRIPT. O armazenamento dos dados do sistema foi realizado pelo banco de dados MySQL, devido a sua compatibilidade e facilidade de comunicação.

Para a comunicação entre a aplicação móvel e o banco de dados foi necessário criar um módulo de comunicação HTTP, onde a aplicação móvel faz a requisição ao módulo de comunicação e o mesmo interpreta a solicitação, faz a consulta no banco de dados e retorna à aplicação móvel. O módulo foi desenvolvido na linguagem PHP e o retorno da requisição, no formato JSON.

2.4.4. Considerações Gerais

Os trabalhos relacionados citados versam sobre sistemas de diferentes âmbitos em diferentes contextos, mas todos trabalham na forma cliente servidor. Com base nesses trabalhos, será desenvolvida uma aplicação *Web Service* baseada na arquitetura SOAP para gerenciamento de frotas de taxis, utilizando-se geolocalização.

2.5. Metodologia FDD

Na estruturação deste trabalho será utilizada a metodologia ágil FDD (*Feature Driven Development*), Desenvolvimento Guiado por Funcionalidades, que é uma metodologia para gerenciamento e desenvolvimento de *software.* Ela combina as melhores práticas do gerenciamento ágil de projetos com uma abordagem completa para Engenharia de Software orientada por objetos, sempre mantendo resultados frequentes, tangíveis e funcionais [FDD 2014].

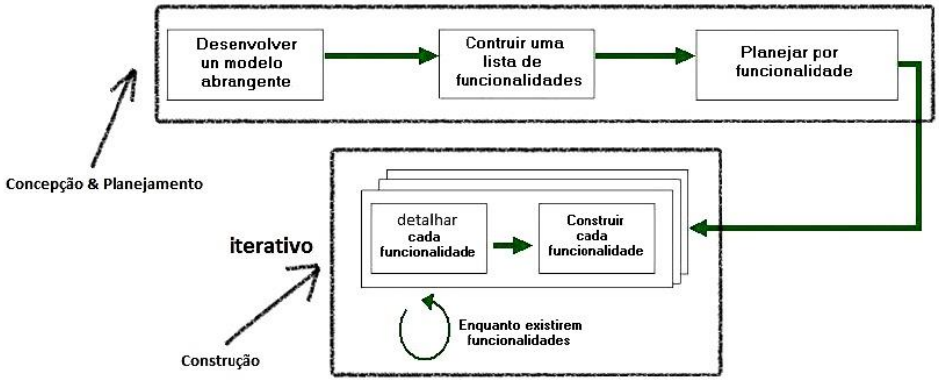

Figura 2. Fases do FDD [Santos 2010].

Na Figura 2 são descritos os cinco processos que compreendem a metodologia FDD [Holl 2011]:

- **Desenvolver um Modelo Abrangente:** consistem em estudos detalhados sobre o domínio do negócio para as partes do produto a serem modeladas, podendo ser em forma de análise de requisitos, orientados por objetos e outras técnicas;
- **Construir a Lista de Funcionalidades:** podendo ser chamado de *product backlog*, consiste na decomposição funcional do domínio, em áreas de negócio, atividades de negócio e funcionalidades;
- **Planejar por Funcionalidades:** planejamento, ordenação e estimativa das atividades a serem implementadas, baseando-se em fatores como carga de trabalho da equipe e complexidade;
- **Detalhar por Funcionalidades:** dentro de uma iteração ocorre o detalhamento dos requisitos e outros artefatos refinando-se os modelos de objetos com a definição de padrões e esqueletos de código;
- **Construir por Funcionalidades:** é realizado o desenvolvimento e em seguida a realização dos testes. Após, ocorre a junção das funcionalidades para preparar o produto de entrega funcional.

3. Desenvolvimento

Nesta seção são elencados os diagramas necessários para o desenvolvimento do trabalho baseado na metodologia FDD.

Observando-se o diagrama de atividade, pode-se ter um entendimento mais apurado do modo de operação do *Web Service*, com base na aplicação de Franco (2013), como mostra a Figura 3, a seguir.

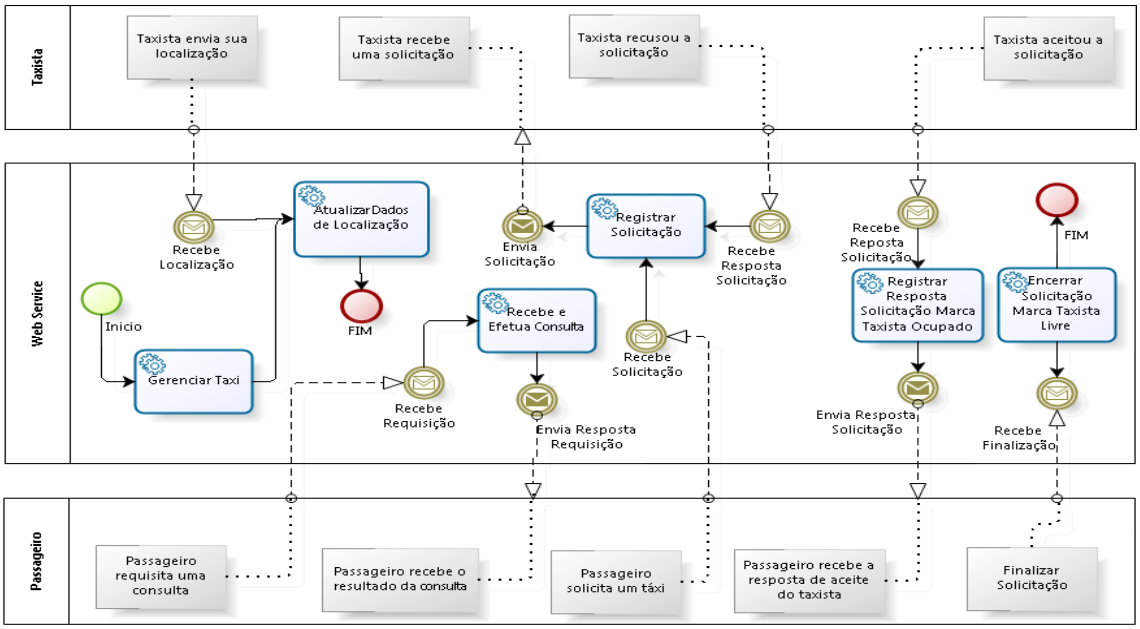

Figura 3. Diagrama de Atividade.

Utilizando a metodologia FDD para guiar o desenvolvimento deste trabalho, tem-se a primeira fase que trata sobre a concepção e o planejamento, os quais foram melhor detalhados no TFG I. No presente trabalho, será abordada a parte de construção do projeto.

3.1. Desenvolver um Modelo Abrangente

No primeiro processo do FDD é necessária a execução de tarefas da formação da equipe de modelagem, explanação da área de conhecimento do projeto, estudo de documentos auxiliares à área de conhecimento em questão, criação do modelo inicial, refinamento e anotações [Cazarotto 2012].

Para este modelo, são identificadas as classes principais do *Web Service* e da aplicação *Android* conforme mostra a Figura 4:

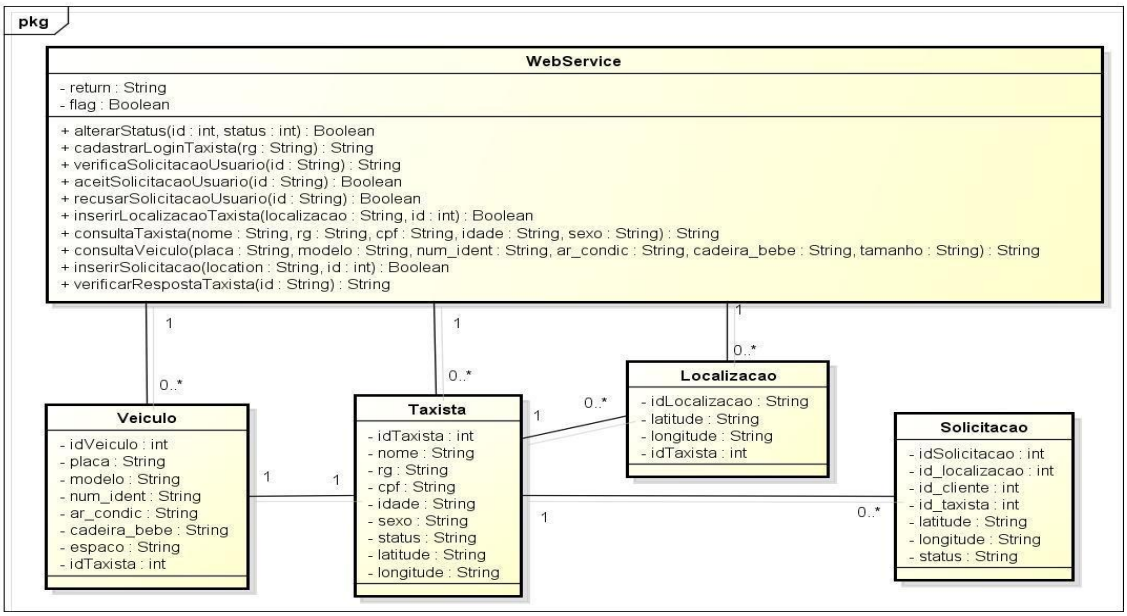

Figura 4. Diagrama de Classes.

Baseado na arquitetura dos sistemas foi elaborado um modelo de Entidade Relacionamento (ER), conforme mostra na Figura 5.

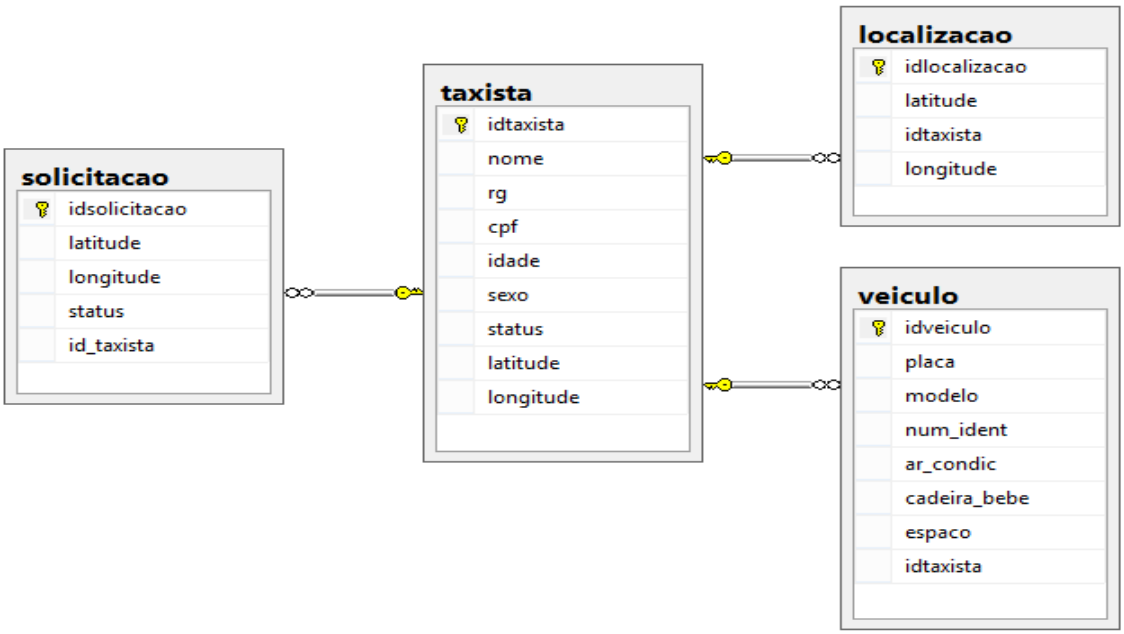

Figura 5. Diagrama Entidade Relacionamento.

3.2. Construir Lista de Funcionalidades

Este processo é uma atividade que abrange todo o projeto, de forma que possam ser identificadas todas as suas funcionalidades para atenderem aos seus requisitos [Magno 2007].

Com base na documentação anterior, os processos são subdivididos em áreas afins e consecutivamente, essas áreas são decompostas em pequenas funcionalidades que foram descritas no TFG I [Schlestein 2014].

3.3. Planejar por Funcionalidade

Neste item, é feito o planejamento, ordenação e a estimativa das atividades que devem ser implementadas, baseados em fatores como a carga de trabalho da equipe e complexidade [Holl 2011].

A segunda fase da metodologia FDD tem como objetivo o foco no desenvolvimento e entrega das funcionalidades de maneira iterativa e incremental. Esta fase de construção divide-se em dois itens descritos a seguir.

3.4. Detalhar por Funcionalidade

Neste processo ocorre o detalhamento dos requisitos e outros artefatos, refinando os modelos de objetos com a definição de padrões e esqueletos de código. A Figura 6 mostra a sequência de execução dos métodos que se referem à aplicação *Android* do taxista com seus respectivos parâmetros e retornos.

Figura 6. Diagrama de Sequência Taxista.

Na Figura 7, mostra-se a sequência de execução dos métodos que se referem à aplicação *Android* do usuário, identificando o fluxo das informações em cada método, recebimento de parâmetros e retorno.

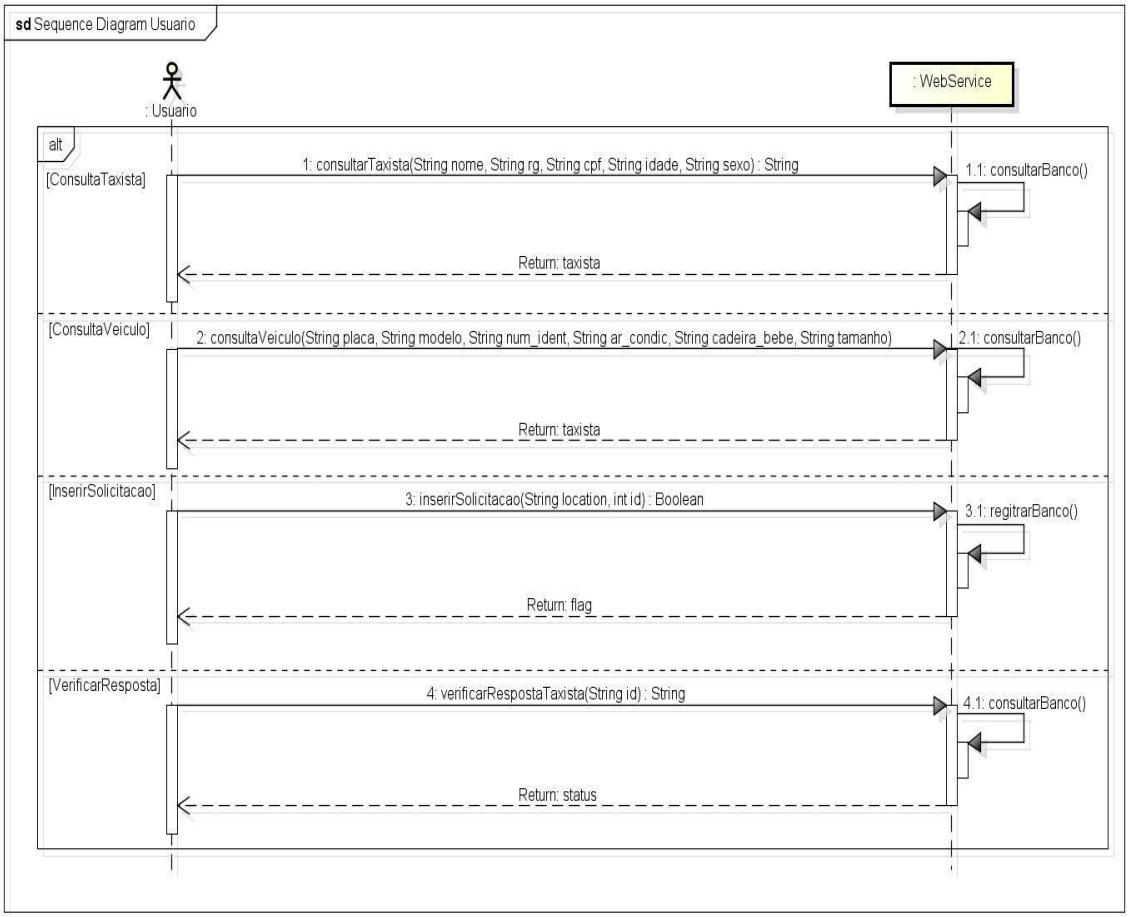

Figura 7. Diagrama de Sequência Usuário.

3.5. Construir por Funcionalidade

Nesta seção, é realizado o desenvolvimento e em seguida, a realização dos testes. Após, ocorre a junção das funcionalidades para preparar o produto de entrega funcional [Holl 2011].

O *Web Service* foi desenvolvido, utilizando-se a linguagem de programação *C#* e, como IDE de programação foi utilizado o *Visual Studio 2015* [Visual Studio 2015]. O banco de dados utilizado para o armazenamento dos dados é o *SQLServer* e, para a integração do banco de dados com o *Visual Studio,* foi utilizado o *ADO.NET Entity Framework* [Entity 2015] que efetua a instanciação das tabelas do banco de dados em classes dentro da aplicação, facilitando as consultas, inserções, atualizações, exclusões e o envio de objetos por parâmetro.

Por meio do *Visual Studio* foram criados os *WebMethods*, que são métodos independentes uns dos outros e cada um é publicado, disponibilizado e acessado individualmente.

Cada *WebMethod* tem total autonomia em relação aos outros serviços sendo responsável pelo seu próprio domínio. O conteúdo de cada serviço possui a codificação necessária para receber os parâmetros enviados pela requisição do cliente, efetuar as consultas e processamentos necessários e retornar ao cliente os dados da requisição.

Na Figura 8 pode-se observar um *WebMethod* que foi implementado no projeto, onde são recebidos cinco parâmetros, é efetuada uma consulta no banco de dados e retornado o resultado da consulta.

```
[WebMethod]
public string consultaVeiculo(string placa, string modelo, string num_ident, string ar_condic, string cadeira_bebe, string tamanho)
{
    #region Declaração
    string retorno = string.Empty;
    Modal01 db = new Modal01();

    var query = (from v in db.veiculo
                 join t in db.taxista on v.idtaxista equals t.idtaxista
                 join l in db.localizacao on t.idtaxista equals l.idtaxista
                 where v.placa.Equals(placa) || v.modelo.Equals(modelo) || v.num_ident.Equals(num_ident)
                 || v.ar_condic.Equals(ar_condic) || v.cadeira_bebe.Equals(cadeira_bebe)
                 || v.espaco.Equals(tamanho) select t).ToList();

    foreach (var item in query) {
        retorno += item.idtaxista.ToString() + "#";
        retorno += item.nome.ToString() + "#";
        retorno += item.rg.ToString() + "#";
        retorno += item.cpf.ToString() + "#";

        foreach (var item2 in item.localizacao)
        {
            retorno += item2.latitude.ToString() + "#";
            retorno += item2.longitude.ToString() + "@";
        }
    }

    return retorno;
    #endregion
}
```

Figura 8. WebMethod.

A troca de mensagens entre o *Web Service* e a aplicação cliente é efetuada com base na arquitetura SOAP. Essa arquitetura é responsável por montar o envelope conforme o padrão SOAP, em formato XML, com os dados necessários e enviá-lo via protocolo HTTP.

Na Figura 9, é exibido um trecho do código onde é montado o envelope, serializado e enviado via HTTP, bem como o retorno do *Web Service*.

```
PropertyInfo pi = new PropertyInfo();
pi.setName("id");
pi.setValue(id_taxista);
pi.setType(String.class);
request.addProperty(pi);

SoapSerializationEnvelope envelope = new SoapSerializationEnvelope(SoapEnvelope.VER11);
envelope.dotNet = true;
envelope.setOutputSoapObject(request);
HttpTransportSE androidHttpTransport = new HttpTransportSE(URL);

try {
    androidHttpTransport.call(SOAP_ACTION, envelope);
    SoapPrimitive result = (SoapPrimitive) envelope.getResponse();
    retorno = result.toString();
} catch (Exception e) {
    Log.i("ConexaoUsuario", e.getMessage().toString());
}
```

Figura 9. Montagem do Envelope SOAP.

O próximo passo do desenvolvimento foi a configuração do servidor IIS (*Internet Information Service*) para a publicação e hospedagem dos serviços do *Web Service*, para posterior comunicação com os clientes via HTTP.

Na aplicação cliente, para ter acesso ao *Web Service* é necessário que o desenvolvedor da aplicação possua a URI (*Unifrom Resource Identifier*) que é o endereço do mesmo, onde assim, terá acesso ao arquivo WSDL o qual possui todas as informações de acesso e os métodos que o compõe.

Na Figura 10, é possível observar como estão configuradas as URIs da aplicação cliente para ter acesso ao método “alteraStatus” do *Web Service*.

```
private static final String NAMESPACE = "http://tempuri.org/";
private static final String URL = "http://10.0.2.2:9090/webservice.asmx";
private static final String SOAP_ACTION = "http://tempuri.org/alteraStatus";
private static final String METHOD_NAME = "alteraStatus";
```

Figura 10. Endereços de Acesso ao Web Service.

Após a conclusão das configurações da aplicação cliente para o acesso ao *Web Service*, o sistema cliente já está apto a consumi-lo.

4. Validação

O presente estudo refere-se ao desenvolvimento de um *Web Service* o qual fará o gerenciamento de frotas de taxis com base na localização e no controle das chamadas. Para validar o *Web Service* foi utilizado o trabalho desenvolvido por Franco (2013) que dispôs de duas aplicações *Android* para o gerenciamento de frotas de táxi.

As duas aplicações *Android* têm uma iteração contínua com o *Web Service* por meio de troca de mensagens XML que trafegam via HTTP. Estas mensagens contém os dados necessários para que as aplicações e o serviço disponibilizado pelo *Web Service* consigam trocar informações para o pleno funcionamento do gerenciador de táxis.

A aplicação do taxista, periodicamente, envia sua localização e verifica se existe alguma solicitação de corrida junto ao *Web Service*. Já, a aplicação do cliente terá iteração com o mesmo, quando o usuário fizer alguma pesquisa ou quando estiver aguardando a resposta do taxista.

O usuário, ao fazer uma pesquisa de filtro pelos dados do taxista ou pelos dados do veículo, faz uma requisição ao *Web Service* para que o mesmo realize uma consulta no banco de dados, identificando todos os taxista que se encaixem na pesquisa efetuada, retornando na tela do seu aplicativo, uma lista dos mesmos, de forma que ele possa selecionar o que melhor se adequar.

Ao escolher o taxista, outra requisição é enviada ao *Web Service* informando qual deles foi escolhido e registrando uma solicitação no banco de dados. O taxista receberá em seu aplicativo, uma mensagem solicitando se ele aceita ou recusa a corrida. Se ele recusar, é enviado ao *Web Service* uma requisição alterando o *status* dele para “livre”, ficando disponível para próximas corridas. Se ele aceitar, é enviado ao *Web Service* uma requisição alterando o *status* dele para “ocupado” e o usuário receberá uma mensagem informando-o que o taxista aceitou a corrida e já tem a possibilidade de visualizar no mapa, o deslocamento dele até o seu local.

Ao final da corrida o taxista finaliza o atendimento e no seu próprio aplicativo altera o seu *status* para “livre” novamente, sendo possível efetuar novas corridas.

5. Conclusão

O *Web Service* desenvolvido neste projeto é um serviço de gerenciamento de frotas de taxis que foi publicado e disponibilizado para gerenciar frotas de taxis por meio da localização e controle de chamadas, indiferente do tipo de *software* e *hardware* que irá

consumir o mesmo, devido a uma das suas principais características que é a interoperabilidade entre aplicações.

A validação do *Web Service* ocorreu por intermédio de uma aplicação *Android* desenvolvida no ambiente de programação *Eclipse* e posteriormente importada no *Android Studio*, o que trouxe alguns contratempos quanto a compatibilidade de versões do SDK do *Android*, erros na instalação e funcionamento do *Android Studio* versão 1.2 e erros na execução do emulador, o que consumiu um tempo grande na tentativa de solucionar esses problemas.

Mesmo após a realização de diversos testes, não foi obtido sucesso no desenvolvimento do *Web Service* na linguagem de programação *Java*, pelo fato de não ocorrer a comunicação entre a aplicação *Android* e o *Web Service*, o que impulsionou a procura de outra plataforma para desenvolvimento, onde foi utilizada a linguagem *C#*.

Foi necessária a instalação do *Visual Studio 2015,* onde também houve problemas na integração com os bancos de dados *MySQL* e *PostgreSQL*, erros na configuração do servidor IIS e erros na importação do *Entity Framework.* Problemas estes que foram solucionados efetuando a atualização da versão do *Android Studio* para a versão 1.4 e utilizando o banco de dados *SQLServer*, sendo assim, foi possível efetivar o funcionamento e a ligação entre as aplicações *Android*, *Web Service* e o banco de dados, tornando o projeto operativo efetuando as trocas de mensagens XML dentro do padrão SOAP e disponível para qualquer aplicação indiferente de sua arquitetura que necessite de gerenciamento de frota.

Um item a ser melhorado em trabalhos futuros é tornar o *Web Service* mais genérico para que outros sistemas também possam utilizar suas funcionalidades, pois atualmente, ele está direcionado à aplicação de Franco (2013) para a validação do projeto.

Outro ponto bastante importante a ser estudado e que não foi possível implementar em tempo hábil, devido ao grande números de contratempos e erros nas aplicações, foi o tratamento dos retornos do *Web Service* em tipos complexos como: objetos ou listas. Atualmente estes retornos são realizados com *strings* para melhor se adequarem ao trabalho desenvolvido por Franco (2013).

6. Referências

Android Studio (2014), “Android Studio 1.0”, http://developer.android.com/tools/studio/index.html. Abril.

Cazarotto, L. P. (2012) “Desenvolvimento de Aplicação Web para Validação de Comprovante de Compra Emitido em Plataforma de Comércio Eletrônico Magento”, Centro Universitário Franciscano – UNIFRA, Santa Maria – RS, p.115.

Dantas, D. C. T. (2007) “Simple Object Access Protocol (SOAP)”, http://www.gta.ufrj.br/grad/07_2/daniel/ , Outubro.

Entity (2015), “Visão geral do Entity Framework”, Microsoft, https://msdn.microsoft.com/pt-br/library/bb399567(v=vs.110).aspx, Outubro.

FDD (2014), “FDD”, http://www.heptagon.com.br/fdd, Abril.

Ferreira, C. de F.; Mota R. D. (2014), "Comparando Aplicação Web Service REST e SOAP", UNIPAR – Universidade Paranaense, http://web.unipar.br/~seinpar/2014/artigos/pos/Cleber_de_F_Ferreira_Roberto_Dias_Mota%20(1).pdf, Outubro.

Ferreira, F. (2014), "Prós e Contras da Geolocalização", http://www.gpspoint.com.br/gps/88-pros-e-contras-da-geolocalizacao, Abril.

Fincotto, M. A.; Santos, M. T. P. (2014), "Automação Comercial Utilizando Dispositivos Móveis – Um Foco na Plataforma Android", Departamento de Computação – UFSCar, São Carlos – SP.

Franco, J. R. (2013), "Sistema para Localização e Gerenciamento de Táxis", Centro Universitário Franciscano – UNIFRA, Santa Maria – RS, p.50.

Gomes, D. A. (2014), "Web Services SOAP em Java: Guia prático para o desenvolvimento de web services em Java", http://novatec.com.br/livros/soapjava2ed/capitulo9788575223567.pdf, Outubro.

Holl, R. (2011), "Metodologia Ágil de Desenvolvimento de Software FDD", https://htwojsystem.wordpress.com/2011/06/26/metodologia-agil-de-desenvolvimento-de-software-fdd/, Abril.

Isernhagen, C. C. dos S. (2014), "Sistema para Controles de Horários em Quadras Esportivas", Centro Universitário Franciscano – UNIFRA, Santa Maria – RS, p.16.

Lecheta, R. R. (2010), "Google Android: Aprenda a criar aplicações para dispositivos móveis com o Android SDK". 2. ed. São Paulo: Novatec Editora.

Linux (2006), "O que é Linux", http://br-linux.org/2008/01/faq-linux.html, Abril.

Lopes, C. J. F.; Ramalho, J. C. (2004), "Web Sevice: Metodologia de Desenvolvimento", http://www4.di.uminho.pt/~jcr/XML/publicacoes/artigos/2004/LR04.pdf, Outubro.

Magno, A. F. (2007), "Um Guia de Rápido Aprendizado para a Feature-Driven Development", São Paulo – SP, http://homes.dcc.ufba.br/~mauricio052/Engenharia%20de%20Software%20I/FDD/FDD%20Em%20Uma%20Casca%20De%20Banana.pdf, Abril.

Medeiros, W. N. de (2014), "Consumindo Web Services em Aplicações Android", DevMedia, http://www.devmedia.com.br/consumindo-webservices-em-aplicacoes-android/26866, Outubro.

Moro, T. D.; Dorneles, C. F., Rebonatto, M. T. (2011), "Web Services WS-* versus Web Services REST", Universidade de Passo Fundo – UPF.

Padoin, E. L.; at al (2007), "Mobilidade e Monitoramento de Subestações Elétricas Através de Serviços Web", UNIJUI, http://www.sirc.unifra.br/artigos2007/artigo14.pdf, Outubro.

Reverbel, F. (2006), "O Que São Web Services", http://www.ime.usp.br/~reverbel/SOD-06/trabalhos/fachada-ws/node2.html, Junho.

Rocha, L. (2014), “Android Studio: Ferramenta de Criação de Apps da Google Ganha Versão 1.0”, http://www.tecmundo.com.br/android/69111-android-studio-ferramenta-criacao-apps-google-ganha-versao-1-0.htm, Abril.

Sant’anna, M. (2015), “SOAP e Web Services”, http://www.linhadecodigo.com.br/artigo/38/soap-e-webservices.aspx, Maio.

Santos, F. N., at al. (2010), “FDD – Feature Driven Development”, http://pt.slideshare.net/engenhariadesoftwareagil/fdd-5139226, Abril.

Schlestein, L. A. (2014), “Protótipo de Controle de Tráfego Utilizando Arduino”, Centro Universitário Franciscano – UNIFRA, Santa Maria – RS, p.21.

Ushisima, R. (2011), “Consumindo Web Services em Aplicações Android”, http://zbra.com.br/2011/03/30/consumindo-web-service-em-aplicacoes-android/, Abril.

Visual Studio (2015), “Visual Studio 2015”, Microsoft, https://www.visualstudio.com/pt-br/vs-2015-product-editions.aspx, Outubro.