UNIVERSIDADE FEDERAL DO PAMPA

Madson Verdi Junior

UMA FERRAMENTA PARA SINCRONIZAÇÃO DE CONTEÚDOS PRODUZIDOS PELO SOFTWARE QGIS

Alegrete
Novembro de 2017

Madson Verdi Junior

UMA FERRAMENTA PARA SINCRONIZAÇÃO DE CONTEÚDOS PRODUZIDOS PELO SOFTWARE QGIS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Orientador: Claudio Schepke

Alegrete
Novembro de 2017

Madson Verdi Junior

UMA FERRAMENTA PARA SINCRONIZAÇÃO DE CONTEÚDOS PRODUZIDOS PELO SOFTWARE QGIS

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Ciência da Computação da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Ciência da Computação.

Trabalho de Conclusão de Curso defendido e aprovado em 27 de novembro de 2017
Banca examinadora:

Claudio Schepke
Orientador
Unipampa

Andrea S. Bordin

Andrea Sabedra Bordin
Unipampa

Jean Felipe Patikowski Cheiran
Unipampa

AGRADECIMENTOS

Agradeço primeiramente a minha família que forneceu apoio nas horas de maior dificuldade.

Ao professor orientador Claudio Schepke pelos conhecimentos transmitidos e paciência durante a elaboração deste trabalho, muito obrigado.

RESUMO

O desenvolvimento de um projeto que utiliza Sistemas de Informações Geográficas (SIGs) pode envolver vários usuários, fazendo com que os usuários necessitem compartilhar a informação trabalhada entre os integrantes deste projeto. Para o compartilhamento de informações existem inúmeras alternativas, onde as principais alternativas disponíveis atuam de forma centralizada e estão disponíveis nos principais sistemas SIGs. Algumas das soluções para compartilhamento de informações consiste em centralizar toda a informação em um banco de dados. Utilizando como base para este trabalho a utilização de SIGs, bancos de dados geográficos, além de algumas definições e técnicas de Engenharia de Software (ES). A metodologia contou com a aplicação de análise e revisão de requisitos, uma subárea da ES, para levantar e verificar os requisitos para execução da implementação desta ferramenta. Foi definido que sua implementação contaria com duas fases: A primeira fase voltada para a validação da ideia e definição de funcionalidades básicas necessárias, a implementação de um `Servidor` e um teste funcional; a segunda fase voltada para a implementação de um *plugin* para o SIG QGIS. Os resultados obtidos neste trabalho estão divididos em duas grandes categorias: **análise de requisitos** e **implementação**, onde a implementação foi subdividida em duas fases. Durante a análise de requisitos foi possível identificar as funcionalidades que a ferramenta deveria possuir. A ferramenta é composta por um `Cliente`, um *plugin* para o Software QGIS e um `Servidor`, que armazena todas as informações geradas pelos usuários através do QGIS. A primeira fase da implementação foi composta pela configuração de um servidor com o banco de dados PostgreSQL e o plugin PostGIS e o desenvolvimento de um `Cliente` capaz de realizar as operações de sincronização a fim de validar a ferramenta, onde foi avaliado o comportamento da ferramenta na sincronização. Com a ferramenta validada foi possível prosseguir para a segunda fase da implementação, que contou com a Prototipação do *plugin*, o desenvolvimento do *plugin* e a realização de testes de usabilidade. Os protótipos foram gerados utilizando o software *Pencil Project*, e forneceram uma visão de como o *plugin* deveria se comportar. A implementação do *plugin* foi realizada tomando como base a implementação da primeira fase e os protótipos de tela gerados nesta fase. Através dos testes de usabilidade realizados foi possível identificar um problema crítico, que fazia a ferramenta parar de funcionar. A informação após ser transmitida para o `Servidor` fica armazenada em um banco de dados. Esta informação pode ser trabalhada de inúmeras maneiras, mas a mais observada, principalmente na análise de requisitos está relacionada a divulgação destas informações através dos padrões WMS ou WTMS, deixando a informação, ou parte dessa informação disponível para acesso *HTTP*.

Palavras-chave: SIG (Sistema de Informações Geográficas), banco de dados geográficos, Software QGIS, PostGIS, Sincronização de dados.

ABSTRACT

A Geographic Information System (GIS) project development can involve multiple users. The users need to share the data information among the members of the project. For data sharing There are some alternatives for data sharing. The main available alternatives manage centrally the data an are available in the main GISs systems. Some of data sharing solutions concentrate the information in a data base. We use GISs, geographic DBs, as well some definitions an techniques of software engineer (SE) as concepts to develop a distributed solution. In the methodology we apply an analysis and review of requirements, sub-area of SE, to establish and verify the requirements analysis and revision for implementation of this tool. The implementation was defined in two phases: the first was leaded to the idea of validation and basic functions definitions necessary, the implementation of a server and a functional test; the second phase was leaded to the implementation of a plugin for QGIS. The obtained results in this work was divided in two categories: requirement analysis and implementation. The implementation was subdivided in two phases. During the requirement analysis it was possible to identify the main functions needed by the tool. The tool is composed by a Client, a plugin for QGIS and a Server, that store all the generated informations of the QGIS users. The first phase of the implementation was composed by a server configuration with PostgreSQL data base and the QGIS plugin and the development of a Client able to made synchronizarions operations. We validate the tool and the behavior of this tool. After the validation, it was able to proceed to phase two. This phase was formed by plugin prototipation, plugin development and the usability tests. The generated prototypes was made using Pencil Project software, and provide an abstract vision of how the plugin could to work. The plugin implementations was based on the implementation of first phase and the generated prototypes. Throw the executed usability tests it was possible to identify a critical problem, that made the tool to stop. All the information sent to the server stay stored in a data base. This stored information can be worked in many ways, but the most observed, especially in the requirements analysis, is related to the disclosure of this information through the WMS or WTMS standards, leaving the information or part of that information available for access HTTP.

key-words: GIS (Geographic Information System), geospatial data base, QGIS Software, PostGIS, Data Synchronization.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

BD banco de dados

CAD Computer Aided Design e Computer Aided Drafting

CGI Common Gateway Interface

CP Concurrent Probing

CTA *Concurrent Think Aloud*

ES Engenharia de Software

GPS *Global Positioning System*

HTTP Hypertext Transfer Protocol

IEEE Institute of Electrical and Electronics Engineers

OGC Open Geospatial Consortium

QGIS Software QGIS

RP *Retrospective Probing*

RTA Retrospective Think Aloud

SGBD Sistema de Gerência de Banco de Dados

SIG Sistema de Informações Geográficas

SQL *Structured Query Language*

VM *Virtual Machine* ou Máquina Virtual

WMS *Web Map Service Interface Standard*

WTMS *Web Map Tile Service Implementation Standard*

XML *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

O desenvolvimento de um projeto que utiliza Sistemas de Informações Geográficas (SIGs) pode envolver vários usuários, fazendo com que os usuários necessitem compartilhar a informação trabalhada entre os integrantes deste projeto.

Para o compartilhamento de informações existem inúmeras alternativas, onde as principais possibilidades disponíveis atuam de forma centralizada e estão disponíveis nos principais sistemas SIGs. Algumas das soluções para compartilhamento de informações consistem em centralizar toda a informação em um banco de dados. Outras alternativas buscam centralizar essa informação em servidores de dados compartilhados em uma rede local, onde possa ser acessado por todos os usuários envolvidos na maior parte do tempo. Porém nas duas principais alternativas apresentadas é necessário estar conectado, de algum modo, a um servidor, seja ele de banco de dados ou de arquivos, deixando o usuário limitado a um ambiente específico para realização do trabalho. Com base nas informações apresentadas é possível identificar que os sistemas atuais são restritos a determinados modos de uso, seja utilizando um servidor para armazenamento das informações ou trabalhando de forma isolada, onde cada usuário tem seus dados locais, mas não as duas formas. Este trabalho busca apresentar uma forma de integrar os benefícios da centralização das informações em um servidor de banco de dados com dados locais, deixando os usuários menos dependentes da estrutura do servidor para realizar o trabalho.

Para abordar este assunto de forma sistemática e buscando um fácil entendimento para o leitor, este trabalho foi subdividido da seguinte maneira:

Inicialmente uma abordagem da problemática tratada está disponível no Capítulo 1. Logo após são apresentadas as fundamentações teóricas necessárias para um melhor entendimento do trabalho. Este conteúdo pode ser observado no Capítulo 2. A metodologia é apresentada no Capítulo 3 e são abordados todos os processos relativos ao desenvolvimento deste trabalho. Após a fundamentação e metodologia, os Resultados são apresentados no Capítulo 4 e contam com todo o conteúdo e dificuldades enfrentadas na implementação da ferramenta. Para concluir este trabalho são apresentadas as considerações finais e trabalhos futuros no Capítulo 5.

1.1 Objetivo geral

Este trabalho busca definir um método objetivo e de fácil manutenção para a centralização das informações geradas por usuários que utilizam SIGs em seus projetos, deixando esta informação acessível a todos interessados nela.

1.2 Objetivos específicos

Este trabalho consistirá no desenvolvimento de uma ferramenta capaz de atender o Objetivo geral. Para isso alguns objetivos foram definidos, são eles:

- definir por quem a ferramenta será utilizada;
- definir como a ferramenta será utilizada;
- definir a metodologia de desenvolvimento;
- definir a metodologia para avaliação da ferramenta;
- publicar a ferramenta.

2 FUNDAMENTAÇÃO TEÓRICA

2.1 Sistemas de Informações Geográficas

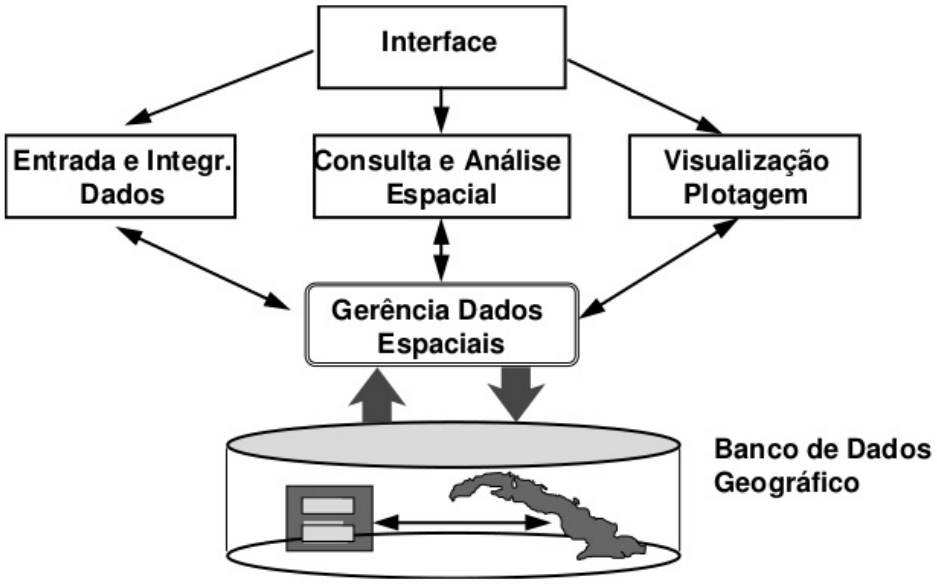
Sistema de Informações Geográficas (SIG) é o termo aplicado a sistemas que realizam o tratamento computacional de dados geográficos. Diferentemente de um sistema de informações convencional um SIG é capaz de armazenar além das descrições necessárias, as geometrias dos diferentes tipos de dados geográficos. Com base nestes conceitos, é possível indicar as principais características dos SIGs:

- inserir e integrar, numa base de dados, informações espaciais provenientes de meio físico-biótico, de dados censitários, de cadastros urbano e rural, e outras fontes de dados como imagens de satélite, e GPS;

- oferecer mecanismos para combinar várias informações, através de algoritmos de manipulação e análise, bem como consultar, recuperar e visualizar o conteúdo da base de dados geográficos.

A Figura 1 ilustra como é realizada a interação com um sistema SIG, na qual a Interface é onde o usuário interage com a aplicação. Esta por sua vez, transforma a informação e encaminha ela para o SGBD que armazena ou retorna a informação. Da mesma forma como um editor de textos é utilizado para manipular o conteúdo de um arquivo de texto, uma aplicação SIG é capaz de manipular dados geográficos.

Figura 1: Arquitetura de Sistemas de Informações Geográficas.



Fonte: Câmara (2005, p. 4)

2.1.1 Dados espaciais e geográficos

Dado espacial é qualquer tipo de dado que descreve fenômenos aos quais esteja associada alguma dimensão espacial. Dados geográficos ou georreferenciados são dados espaciais em que a dimensão espacial está associada à sua localização na superfície da terra, num determinado instante ou período de tempo(CÂMARA, 1996).

Basicamente os dados espaciais estão divididos em dois tipos:

- **Dados de projeto auxiliado por computador (CAD)**: conforme descrito por Ricarte (1987, p.23), a sigla CAD pode ser denominada como *Computer Aided Design e Computer Aided Drafting* e estes dados representam dados espaciais de como objetos são construídos;

- **Dados geográficos**: Representam os mapas rodoviários, de uso da terra, levantamento topográfico etc. A grande diferença de um dado geográfico dos demais dados é ele ser considerado um componente espacial, o que faz com que muitas vezes ele seja referenciado apenas como um dado espacial. Os dados geográficos representam, como foi citado anteriormente, os mapas rodoviários, de uso da terra, levantamento topográfico etc. tendo por base suas coordenadas. Assim é possível a análise do espaço geográfico, é possível, por exemplo, determinar por quais municípios são cortados por uma determinada rodovia.

As características espaciais informam a posição geográfica do fenômeno e sua geometria. As características não-espaciais descrevem os fenômenos e as características temporais informam o tempo de validade dos dados geográficos e suas variações sobre o tempo. A representação espacial de uma entidade geográfica é a descrição da sua forma geométrica associada à posição geográfica.

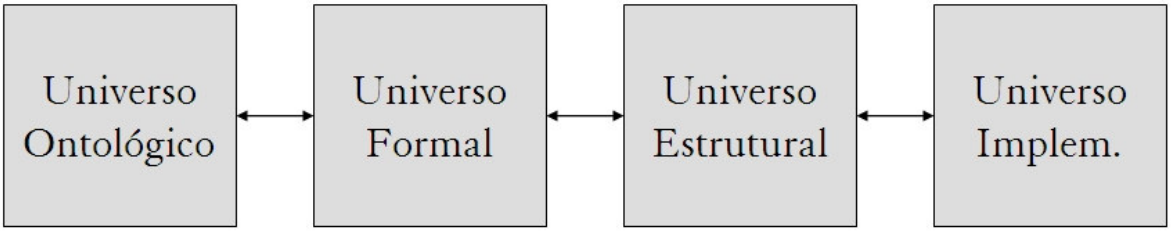
Os dados geográficos possuem propriedades geométricas e topológicas. As propriedades geométricas são propriedades métricas. A partir de feições geométricas primitivas, tais como pontos, linhas e polígonos, os quais representam a geometria das entidades, são estabelecidos os relacionamentos métricos. Esses relacionamentos expressam a métrica das feições com referência a um sistema de coordenadas. De acordo com a geometria são estabelecidas algumas propriedades geométricas tais como, comprimento, sinuosidade e orientação para linha; perímetro e área da superfície para polígonos, volume para entidades tridimensionais, e forma e inclinação tanto para linhas quanto para polígonos (LAURINI; THOMPSON, 1992).

2.1.2 Representação computacional dos dados geográficos

Um dos principais problemas que envolvem a geoinformação é a representação computacional de dados geográficos. Para isso é interessante abordar o paradigma dos

quatro universos, Figura 2, proposto por Gomes e Velho (1995) e posteriormente adaptado para a geoinformação por Câmara (1995).

Figura 2: Paradigma dos quatro universos.



Fonte: Câmara (2005, p. 4)

O universo ontológico é onde são definidas as classes de entidades necessárias, como os tipos de solo, elementos de cadastro urbano, e caracterização das formas do terreno.

O próximo universo inclui os modelos lógicos e as lógicas matemáticas que generalizam o universo ontológico.

O terceiro universo (Universo Estrutural) converte os dados coletados para estruturas de dados geométricas e alfanuméricas, onde o Sistema de Gerência de Banco de Dados é incluído.

O último universo, Universo de Implementação, é onde o sistema é implementado, estas implementações podem ser: geometria computacional, métodos de acesso e processamento de consultas.

2.1.2.1 Teoria da medida

Para representar dados geográficos no computador deve ser descrita sua variação no espaço e no tempo. O processo de medida consiste em associar números ou símbolos a diferentes ocorrências de um mesmo atributo, para que a relação dos números ou símbolos reflita a relação entre as ocorrências mensuradas. As escalas das medidas, segundo Stevens (1946), são: nominal, ordinal, intervalo e razão.

Com base na Tabela 1 é possível observar onde cada tipo de escala pode ser aplicado. Onde a **nominal** descreve um dado, como tipo de solo, vegetação etc. A **ordinal** define em que ordem está, como classes de declividade e aptidão de uso. A escala de **intervalo** pode ser aplicada para definir a distância entre pontos, assim também podendo ser utilizada para medir alturas e por fim. Por fim a escala de **razão**, definindo os valores absolutos, e como exemplo para esta temos: renda, população, taxa de natalidade etc.

Tabela 1: Escalas das medidas.

Scale	**Basic Empirical Operations**	**Mathematical Group Structure**	**Permissive Statistics (invariantive)**
Nominal	Determination of equality	Permutation group	Number of cases Mode Contingency correlation
Ordinal	Determination of greater or less	Isotonic group	Median Percentiles
Interval	Determination of equality of interval or differences	General linar group	Mean Standard deviation Rank-order correlation Product-moment correlation
Ratio	Determination of equality of ratios	Similarity group	Coefficient of variation

Fonte: Stevens (1946, p.678)

2.2 PostgreSQL

PostgreSQL é um banco de dados objeto-relacional e baseado no POSTGRES, versão 4.2, e foi desenvolvido na *University of California at Berkeley Computer Science Department* pelo professor Michael Stonebraker. PostgreSQL é um projeto *open-source* e suporta uma gama enorme de funções SQL.

O nome *Postgres* é derivado de um sistema de banco de dados relacional pioneiro, Ingres. PostgreSQL admite SQL92 e SQL:1999 e oferece recursos como consultas complexas, chaves estrangeiras, triggers, views, integridade transacional e controle de concorrência de múltiplas versões, além de permitir que seja estendido com novos tipos de dados, funções, operadores ou métodos de índice, além de funcionar juntamente com diversas linguagens de programação como: *C*, *C++*, *Java*, *Perl*, *Tcl* e *Python*.

2.2.1 Sistema de tipos do PostgreSQL

O PostgreSQL possui suporte para tipos não usuais o que o torna útil em determinadas aplicações que exigem tipos de dados únicos, para a geração de um novo tipo o comando *create type* deve ser executado. Os tipos que a linguagem atende são:

- **Tipos básicos**: Também conhecidos como tipos de dados abstratos, ou seja, os módulos que encapsulam estado e um conjunto de operações. Estes são implementados abaixo do nível SQL, geralmente em linguagem *C*, onde cada tipo básico vem acompanhado por um tipo array que permite armazenar arrays de tamanhos variáveis do tipo básico vinculado.

- **Tipos compostos**: São uma lista de nomes de campo e seus respectivos tipos básicos, correspondem a linhas de tabela, são criados implicitamente quando uma tabela é criada, mas pode ser definido de forma explicita.

- **Domínios**: Semelhante ao tipo básico, mas contendo restrições sobre os valores permitidos.

- **Pseudotipos**: Podem ser utilizados como tipos de argumento e retorno das funções definidas pelo usuário e admite os seguintes pseudotipos: *any*, *anyarray*, *anyelement*, *cstring*, *internal*, *language-handler*, *record*, *trigger* e *void*.

- **Tipos polimórficos**: Os tipos *anyarray* e *anyelement* são conhecidos, também, como **tipos polimórficos**, onde as funções com argumentos deste tipo são denominadas **funções polimórficas**, as quais podem operar sobre qualquer tipo real.

- **Tipos fora do padrão**: "Os tipos de dados geométricos (*point*,*line*, *lseg*, *box*, *polygon*, *path*, *circle*) são usados em sistemas de informações geográficas para representar objetos espaciais bidimensionais, como pontos, segmentos de linha, polígonos, caminhos e círculos. Diversas funções e operadores estão disponíveis no PostgreSQL para realizar diversas operações geométricas, como escala, tradução, rotação e determinação de intersecções. Além do mais, PostgreSQL admite indexação desses tipos usando árvores-R [1]" Silberschatz et al. (2006, p.656)

- **Linguagens procedurais** A elaboração de funções e a procedimentos armazenados pode ser realizada utilizando diversas linguagens de programação e são registradas por demanda e podem ou não serem consideradas confiáveis, quando uma linguagem é chamada de **não confiável**, ela permite acesso ilimitado ao Sistema de Gerência de Banco de Dados e ao sistema de arquivos, e a escrita de informações armazenadas nela requer privilégios de **superusuário**.

2.2.2 PostGIS, uma extensão para o banco de dados PostgreSQL

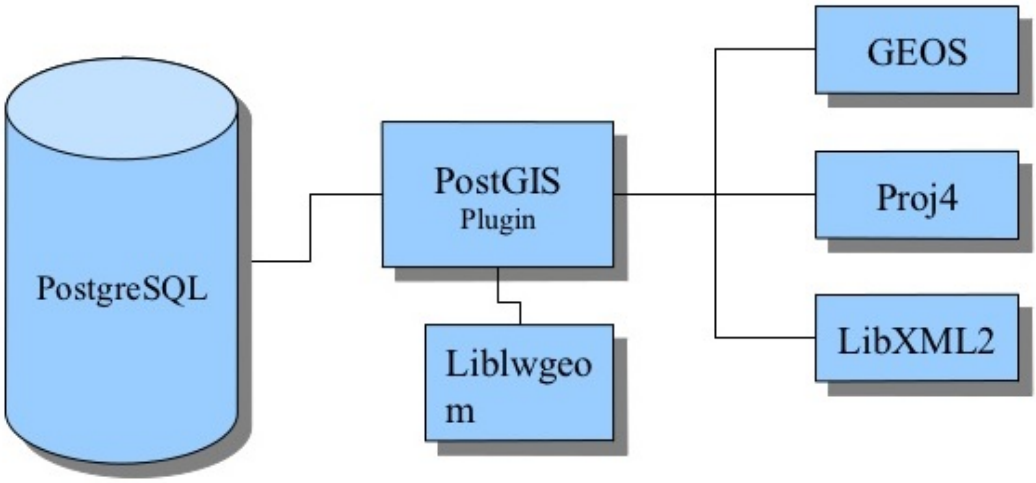
PostGIS foi desenvolvido por *Refractions Research Inc*, como um projeto de banco de dados geográficos. *Refractions* é uma empresa de consultoria Sistema de Informações Geográficas e banco de dados em geral, especializada em integração de dados e desenvolvimento de software personalizado e situada em *Victoria, British Columbia, Canada*, a empresa busca apoiar e desenvolver o PostGIS para melhorar sistemas SIGs, incluindo suporte completo OpenGIS, avançadas construções topológicas (coberturas, superfícies, redes), ferramentas de interface de usuário de *desktop* para visualização e edição de dados de um SIG, e ferramentas de acesso baseadas na web.

[1] Uma árvore-R é uma estrutura de árvore balanceada com os objetos indexados armazenados nos nós folhas, semelhante à árvore B+. Silberschatz et al. (2006, p.617)

O PostGIS funciona como uma extensão para o PostgreSQL, permitindo que o PostgreSQL seja utilizado como banco de dados geográfico, e possui duas funções que auxiliaram no desenvolvimento desta aplicação, `shp2pgsql`[2], que na Tabela 2 tem suas opções exclusivas descritas, e `pgsql2shp`[3] que possui suas opções exclusivas descritas na Tabela 3, as opções comuns para os dois comandos são descritas na Tabela 4.

A arquitetura do PostGIS pode ser visualizada na Figura 3, a camada ***PostGIS Plugin*** representa a espacial, onde as funções e tipos espaciais são carregados, estes tipos e funções são descritos na subseção 2.1.1. ***Liblwgeom*** é uma sub-biblioteca que lida com operações de baixo nível não diretamente ligados ao banco de dados. ***GEOS***, uma biblioteca C++ que implementa alguns algoritmos espaciais. ***Proj4*** está ligado às operações em tempo real. A última camada a ser descrita é a ***LibXML2*** que manipula as entradas de geometrias no formato XML.

Figura 3: Arquitetura do PostGIS, uma extensão para o banco de dados PostgreSQL



Fonte: Ramsey et al. (2005)

2.2.2.1 Comandos shp2pgsql e pgsql2shp do PostGIS

O comando `shp2pgsql` possui a função de realizar a conversão entre um arquivo no formato `.shp` para um arquivo no formato `.sql`, conforme o modelo:

```
1 shp2pgsql [OPTIONS] shapefile [schema.]table
```

Já o comando `pgsql2shp` realiza a conversão inversa, convertendo um arquivo no formato `.sql` para um arquivo no formato `.shp`:

[2] Descrito em: subseção 2.2.2.1
[3] Descrito em: subseção 2.2.2.1

```
pgsql2shp [OPTIONS] database [schema.]table pgsql2shp [OPTIONS] database query
```

Tabela 2: Opções para o comando `shp2pgsql`

Parâmetro	**Descrição**
-s from_srid:to_srid	If -s to_srid 1 is not specified then from_srid is assumed and no transformation happens.
(-d|a|c|p)	These are mutually exclusive options:
-d	Drops the table, then recreates it and populates it with current shape file data.
-a	Appends shape file into current table, must be exactly the same table schema.
-c	Creates a new table and populates it, default if you do not specify any options.
-p	Prepare mode, only creates the table.
-D	Use postgresql dump format (defaults to sql insert statments).
-e	Execute each statement individually, do not use a transaction. Not compatible with -D.
-i	Use int4 type for all integer dbf fields.
-I	Create a GiST index on the geometry column.
-p port	Allows you to specify a database port other than the default (Defaults to 5432).
-u user	Connect to the database as the specified user.
-W	encoding The character encoding of Shape's attribute column (default : "UTF-8").
-N	
-n	policy Specify NULL geometries handling policy (insert,skip,abort).
-G	Only import DBF file.
-T	Use geography type instead of geometry (requires lon/lat data) in WGS84 long lat (-s SRID=4326).
-X	Specify the tablespace for the new index.

2.3 Software QGIS

QGIS trata-se de um Sistema de Informações Geográficas livre e gratuito e é concedida a permissão para copiar, distribuir e / ou modificar este documento sob os termos da Licença de Documentação Livre GNU, Versão 1.2 ou qualquer versão posterior publicada pela Fundação de Software Livre; sem Secções Invariantes, sem textos de Capa e sem contracapa. Surgiu com o propósito básico dos softwares livres, o de compartilhamento livre entre a comunidade.

O QGIS, como em outros Sistemas de Informações Geográficas, é capaz de mostrar mapas em camadas, estas geralmente armazenadas em arquivos e/ou banco de dados,

Tabela 3: Opções para o comando `pgsql2shp`

Parâmetro	**Descrição**
-b	Use a binary cursor.
-f filename	Use this option to specify the name of the file to create.
-h hostname	Specify db server host name defaults to localhost.
-P password	Connect to the database with the specified password.
-r	Raw mode. Do not unescape attribute names and not skip the 'gid' attribute.
-S	Generate simple geometries instead of MULTI geometries.
-w	Use wkt format (for postgis-0.x support - drops M - drifts coordinates).
-m filename	Remap identifiers to ten character names. The content of the file is lines of two symbols separated by a single white space.

Tabela 4: Opções para os comandos `shp2pgsql` e `pgsql2shp`

Parâmetro	**Descrição**
-g geometry_column_name	Specify the name of the geometry column to be (S) created (P) exported.
-k	Keep postgresql identifiers case.
-?	Display this help screen.

P = pgsql2shp; S = shp2pgsql

quando utilizado banco de dados ele opera sobre o PostGIS, e a cada iteração do usuário o banco de dados é atualizado, porém quando o usuário operar fora da rede onde o banco de dados está ou em uma rede que não provê acesso a este banco de dados, não é possível atualizar os mapas que entram em modo de **somente leitura**. Se o QGIS for utilizado com arquivos o problema anteriormente citado não é encontrado.

O sistema também possui a opção de servidor, onde, através de uma aplicação separada é denominada QGIS Server, pode atuar como servidor, fazendo com que os dados armazenados em um banco de dados que este servidor utilizará possa ser utilizado para consultas externas.

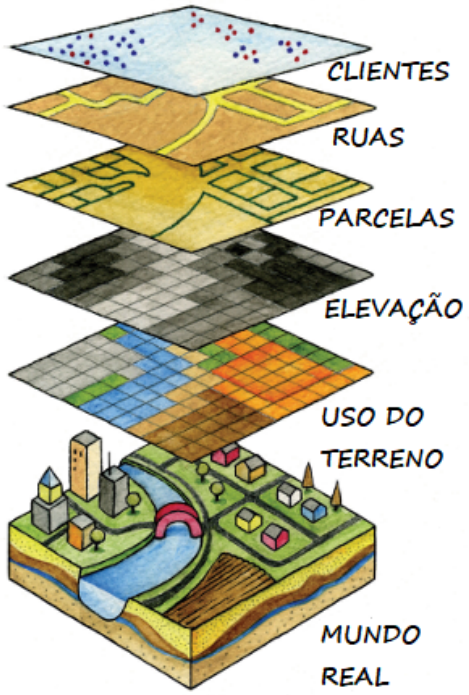
2.4 Open Geospatial Consortium

Fundado em 1994, o *Open Geospatial Consortium* (OGC) é um consórcio da indústria internacional, formado por 514 companhias, governos e universidades unidos para desenvolverem padrões para publicação de conteúdos. Os padrões definidos pelo OGC buscam tornar os dados complexos acessíveis a todos os tipos de aplicações. A gera-

ção destes dados se dá seguindo padrões definidos pelo OGC. Os padrões que se fazem necessários a esta etapa do projeto são WMS[4] e WTMS[5].

A Figura 4 exemplifica a aplicação do padrão WMS onde as camadas, **CLIENTES**, **RUAS**, **PARCELAS**, **ELEVAÇÃO** e **USO DO TERRENO** representam os dados carregados de forma dinâmica, sobre um mapa real, representado no exemplo como a última camada, **MUNDO REAL**.

Figura 4: Exemplo do padrão *Web Map Service Interface Standard*



Fonte: Campbell (2015, p.39)

Já a Figura 5 abstrai o padrão *Web Map Tile Service Implementation Standard* (WTMS), onde blocos podem ser carregados em diferentes níveis de detalhamento. Cada bloco de imagem pode conter as informações de todas camadas, já carregados em forma de imagens, essas divididas em pequenos pedaços.

Um exemplo de utilização bastante popular, que utiliza a união dos dois padrões, WMS e WTMS, é o aplicativo disponível para acesso em navegadores, o Google Maps, que exibe os mapas, em sua visão *Earth* carrega parte de seu conteúdo utilizando o WTMS e

[4] O OpenGIS ®*Web Map Service Interface Standard* (WMS) proporciona uma interface HTTP para requisições de imagens de um ou mais base de dados geográficos, de forma dinâmica, retornando uma imagem em formato digital, renderizada tanto em formatos de imagem (PNG, GIF e JPEG) quanto em formatos vetoriais (SVG e WebCGM). A imagem requerida pode ser transparente e diversas requisições podem serem realizadas a diferentes servidores a fim de cruzar as informações recuperadas.

[5] O OpenGIS ®*Web Map Tile Service Implementation Standard* fornece uma opção mais rápida do que a buscada pelo WMS que retorna exatamente a imagem procurada, ele retorna blocos (*tiles*)) de imagens previamente processadas.

aplica uma nova camada deste com ruas, pontos turísticos, rios etc sobre este utilizando o padrão WMS, conforme Figura 6.

A divulgação do conteúdo armazenado em um banco de dados pode ser realizada através da utilização do QGIS Server.

2.5 Engenharia de Software

O conceito de Engenharia de Software (ES) não é padronizado entre as literaturas, conforme expressado por (PRESSMAN, 2011, p.39) em seu livro, porém alguns conceitos visam definir o que é Engenharia de Software. A primeira tentativa de descrever do que se trata a ES foi dado por (NAUR; RANDELL, 1969):

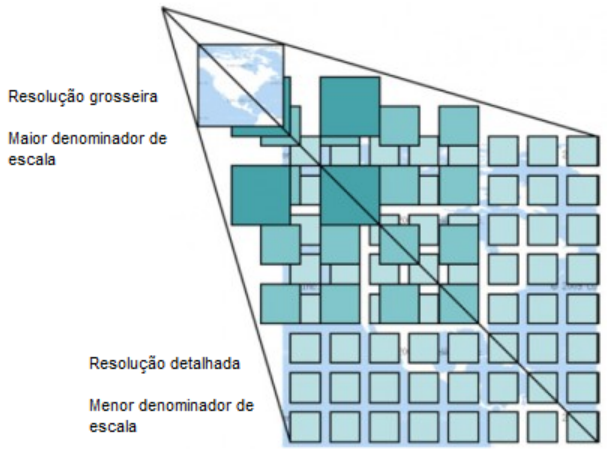
> Engenharia de Software o estabelecimento e o emprego de sólidos princípios de engenharia de modo a obter software de maneira econômica, que seja confiável e funcione de forma eficiente em máquinas reais.

A citação acima despertou uma busca em definir o que é Engenharia de Software e algum tempo depois a IEEE definiu da seguinte forma:

> Engenharia de Software: (1) A aplicação de uma abordagem sistemática, disciplinada e quantificável no desenvolvimento, na operação e na manutenção de software; isto é, a aplicação de engenharia ao software. (2) O estudo das abordagens como definido em (1). (TRIPP, 1994)

Assim, com base na definição ampla citada pela IEEE, áreas específicas da ES foram utilizadas neste trabalho, Análise e revisão de requisitos e Testes de usabilidade.

Figura 5: Exemplo do padrão *Web Map Tile Service Implementation Standard*



Fonte: QGis (2011)

Figura 6: Exemplo do Google Maps

Fonte: ©2016 Landsat,Data SIO,NOAA,U.S. Navy,NGA,GEBCO,Dados do mapa ©2016 Google Maps

2.5.1 Análise e revisão de requisitos

Na área de ES, uma divisão muito importante está relacionada a análise de requisitos, que segundo (PRESSMAN, 2011), esta área define as características operacionais de um software, indicando a interface do software com outros elementos do sistema e estabelece restrições que o software deve atender.

2.5.2 Testes de usabilidade

O teste tem como principal objetivo encontrar erros. O **teste de usabilidade** é um dos vários tipos de testes existentes, este tem como principais objetivos a identificação de problemas relacionados a usabilidade do sistema, coletar dados quantitativos e qualitativos e determinar a satisfação dos participantes com o produto.

A seguir uma lista de benefícios relacionados ao testes de software (USABILITY.GOV, d):

- aprender se os participantes estão habilitados a completar com sucesso as tarefas especificadas;
- identificar quanto tempo demora para que as tarefas especificadas sejam concluídas;
- descobrir o quão satisfeitos os participantes estão com o produto;
- identificar mudanças que melhorem o desempenho e a satisfação do usuário;
- analisar o desempenho para verificar se está de acordo com os objetivos de usabilidade.

Inicialmente deve ser elaborado um **roteiro** para o teste. Este roteiro deve conter o que será realizado, como será realizado, que métricas serão capturadas, a quantidade de participantes e os cenários que serão utilizados. Alguns elementos do teste do roteiro são (USABILITY.GOV, a):

- **Escopo**: Indica o que está sendo testado, como por exemplo, o nome do sistema. Especifica o quanto do sistema será coberto pelo teste (por exemplo: o protótipo, a navegação, navegação e conteúdo etc.).
- **Propósito**: identifica os interesses, questões e objetivos para o teste. Podendo ser bem subjetivos, como por exemplo: "Usuários podem navegar até informações importantes a partir da tela inicial?", ou podem ser bem objetivos, como por exemplo: "Usuários podem encontrar facilmente a barra de busca?". Em cada rodada de testes provavelmente terão muitos objetivos gerais e específicos a serem analisados. Os interesses devem definir quais cenários serão escolhidos no teste de usabilidade.
- **Agendamento e Locação**: Indica quando e onde o teste será realizado. Após a realização do agendamento será necessário especificar quantas sessões serão necessárias no dia e quais seus horários.
- **Sessões**: Necessário a descrição das sessões, sua duração (normalmente 90 minutos). Quando o agendo estiver sendo realizado, é indicado manter um intervalo de aproximadamente 30 minutos entre uma sessão e outra para realizar o *reset* do ambiente.
- **Equipamento**: Indica o tipo de equipamento que será utilizado: desktop, laptop, *smartphone* etc. Se necessário, incluir informações sobre o tamanho de tela necessário, sistema operacional, navegador etc. Também é necessário indicar se o teste será gravado em forma de vídeo, audio ou usando alguma outra tecnologia.
- **Participantes**: Indica o número e o tipo de participantes que serão recrutados. Descreve como os participantes serão selecionados.
- **Cenários**: Indica o número o tipo de tarefas incluídas no teste. Normalmente em 60 minutos são realizados aproximadamente 10 cenários de testes para desktop/laptop e 8 cenários para *smartphones*.
- **Métricas**: Indica as questões que serão enviadas aos participantes, depois de cada conclusão de cenário algumas questões devem ser respondidas: facilidade e satisfação com a tarefa, facilidade geral, satisfação e sugestões.
- **Métricas quantitativas**: Indica o conteúdo quantitativo que será medido no teste (por exemplo: tarefas concluídas, erros, tempo em uma tarefa etc.).

- **Função**: Incluída a lista da equipe que irá participar do teste e a função de cada um. O especialista em teste de usabilidade deve ser um intermediador entre as sessões. A equipe de usabilidade deve fornecer alguém que irá realizar as anotações. Os demais irão atuar como observadores ou também como anotadores.

Para analisar os dados gerados em um testes é necessário definir métricas para este teste, algumas, são (USABILITY.GOV, a):

- **Tarefa completada com sucesso**: Cada cenário requer que o participante obtenha algum conteúdo que será utilizado em uma tarefa comum. O cenários está completamente concluído quando o participante indica que encontrou a resposta ou atingiu o objetivo da tarefa. Em alguns casos questões de múltipla escolha são fornecidas, nestes casos é necessário incluir as questões e suas respostas no roteiro e fornecer elas para os anotadores e observadores.
- **Erros críticos**: Erros que impossibilitam a conclusão de cenários.
- **Erros não-críticos**: Erros que não interferem na conclusão com sucesso da tarefa. Esses erros resultam na conclusão sem a eficiência esperada.
- **Taxa de sem erro**: é o percentual de participantes que concluíram a tarefa sem erro.
- **Tempo na tarefa**: a quantidade de tempo que o participante leva para concluir a tarefa.
- **Recomendações**: com base na resposta e/ou aceitação dos participantes em relação ao produto.

Um teste de usabilidade deve ser executado utilizando um modelo de moderação (USABILITY.GOV, c):

- ***Concurrent Think Aloud* (CTA)**: Utilizado para entender o que os participantes estão pensando enquanto interagem com o produto, os participantes falam em voz alta o motivo que estão realizando determinada tarefa.
- ***Retrospective Think Aloud* (RTA)**: O moderador solicita aos participantes para repassarem os passo no término da sessão.
- ***Concurrent Probing* (CP)**: O moderador solicita aos participantes realizarem tarefas, quando algo interessante é dito ou algo único é realizado, o pesquisador realiza questionamentos sobre a ação.
- ***Retrospective Probing* (RP)**: O moderador realiza os questionamentos sobre as ações dos participantes ao final da sessão.

Após a conclusão do teste de usabilidade os seus resultados devem ser colocados em um relatório onde principalmente as informações pertinentes ao teste devem estar presentes. A análise dos dados deve ser realizada com base nas métricas estabelecidas para a realização das tarefas, uma revisão nas anotações é indicada a fim de encontrar padrões nos problemas enfrentados pelos participantes. A escrita do relatório do teste deve possuir basicamente um resumo, a metodologia, resultados dos testes, conclusões e recomendações.

O relatório, segundo Usability.gov (b), pode possuir elementos gráficos a fim de deixar o leitor do relatório a par do que está sendo testado.

Para um teste de usabilidade possuir algum valor é necessário aprender e implementar melhorias no produto com base nos resultados dos testes.

2.6 Linguagem de programação

Uma boa definição sobre o que é uma linguagem de programação pode ser encontrada em Gudwin (1997), onde o autor explica que para se interpretar um algoritmo em um computador é necessário descrevê-lo de uma forma que o computador esteja apto a executá-lo e esta descrição é realizada através de uma linguagem de programação. No contexto deste trabalho serão utilizadas basicamente duas linguagens de programação, Python[6] e uma Linguagem de banco de dados.

2.6.1 Python

Python é uma linguagem de programação de alto nível, interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte. Foi lançada por Guido van Rossum em 1991. Atualmente possui um modelo de desenvolvimento comunitário, aberto e gerenciado pela organização sem fins lucrativos Python Software Foundation. Apesar de várias partes da linguagem possuírem padrões e especificações formais, a linguagem como um todo não é formalmente especificada. É uma linguagem de propósito geral de alto nível, multi paradigma, suporta o paradigma orientado a objetos, imperativo, funcional e procedural. Possui tipagem dinâmica e uma de suas principais características é permitir a fácil leitura do código e exigir poucas linhas de código se comparado ao mesmo programa em outras linguagens. Devido as suas características, ela é principalmente utilizada para processamento de textos, dados científicos e criação de CGIs para páginas dinâmicas para a web, (ROSSUM; DRAKE, 2017).

Como o Python possui uma documentação e uma comunidade mais ativa, foi utilizado para o desenvolvimento do *Plugin*.

[6] Descrito em: subseção 2.6.1 Python

3 METODOLOGIA

Esta seção trata da metodologia utilizada para o desenvolvimento da ferramenta, onde são descritos as etapas de Análise e revisão de requisitos, descrito em: subseção 2.5.1, a Implementação da solução aborda os passos necessários para a implementação da ferramenta e por último a implementação e execução de testes na seção 3.3 e suas subseções.

3.1 Análise e Revisão de requisitos

Inicialmente foi necessário negociar e levantar os requisitos, conforme descrito por (BOEHM et al., 1998). A negociação é constituída pelas seguintes atividades:

1. identificação dos principais interessados no sistema ou subsistema;
2. determinação das "Condições de ganho" dos interessados;
3. negociação das condições gerais de ganho dos interessados para reconciliá-las em um conjunto de condições "ganha-ganha" para todos os envolvidos.

Com base nos requisitos apresentados acima e a descrição apresentada por (PRESSMAN, 2011), foi possível definir dois itens:

- **Análise de requisitos do problema**: Avaliar quais as tarefas que devem ser realizadas visando definir como proceder com o projeto;
- **Revisão dos requisitos analisados**: Após a análise dos requisitos é necessário avaliar se estes estão todos de acordo, conforme (PRESSMAN, 2011, p. 146-147). Uma revisão nos requisitos pode identificar possíveis problemas a serem enfrentados e também gerar novos requisitos que não foram identificados anteriormente.

3.2 Implementação da solução

O desenvolvimento da ferramenta foi dividido em duas fases:

- Primeira fase:
  - configurar um servidor para armazenar os dados;
  - definir funções, bibliotecas e arquivos binários necessários para transmitir informações para o servidor;
  - definir funções, bibliotecas e arquivos binários necessários para receber informações do servidor;
  - criar um cliente que possa conectar com esse servidor;
  - criar um cliente que possa enviar e receber dados do servidor;

  - implementar testes para validar este cenário.

- Segunda fase:
  - prototipar um *plugin* (ferramenta) para o QGIS;
  - implementar o protótipo gerado utilizando as funções, bibliotecas utilizados na primeira fase;
  - realizar testes de usabilidade do *plugin*.

O desenvolvimento dessa ferramenta, além de ser divida em duas fases, será dividido em duas partes, `Cliente` e `Servidor`. O `Cliente`, ou como está sendo denominado nesta seção, *plugin*, deve funcionar na versões 1.8 ou superior do Software QGIS. O `Servidor` da ferramenta está descrito na subseção 3.2.1. O `Cliente` deverá realizar a sincronização dos dados com o `Servidor`. Além destes detalhes, sua implementação deverá ocorrer utilizando a linguagem de programação Python. A definição da linguagem se deu graças a documentação para desenvolvimento de *plugins* no site do QGIS ser atualizada e possuir uma comunidade ativa.

As etapas para desenvolvimento da primeira fase estão descritas nas seções 3.2.1, 3.2.2 e 3.3.1. Já a segunda fase do desenvolvimento está descrita nas seções 3.2.4, 3.2.5 e 3.3.2.

3.2.1 Servidor da ferramenta

O único requisito para o servidor da ferramenta é possuir o banco de dados PostgreSQL e possuir a extensão PostGIS para tratar os dados geográficos que serão armazenados neste BD. Para habilitar a extensão PostGIS no banco de dados é necessário executar o comando:

```
1 CREATE EXTENSION postgis;
```

Para verificar se a instalação ocorreu com sucesso o comando pode ser executado:

```
1 SELECT postgis_full_version();
```

3.2.2 Geração das consultas em SQL

As consultas devem ser geradas utilizando *Structured Query Language* (SQL) e serem compatíveis com o banco de dados PostgreSQL. Para isso elas devem ser geradas utilizando o comando `shp2pgsql`, que permite gerar os arquivos com a extensão `.sql` à partir de arquivos com a extensão `.shp` utilizados pelo QGIS. A ferramenta deve recuperar todos arquivos, do projeto em que o cliente está trabalhando, com a extensão `.shp`, e aplicar este comando em cada um deles. Os arquivos gerados devem ser armazenados em um ou mais arquivos temporários.

A descrição deste comando está descrita na subseção 2.2.2.1 (Comandos shp2pgsql e pgsql2shp do PostGIS).

3.2.3 Implementação do Cliente da primeira fase

O `Cliente` abordado nesta etapa será a parte da ferramenta que ficará no computador do usuário e irá realizar a comunicação com o servidor. Sua implementação foi realizada seguindo os seguintes componentes:

- **script**: A implementação do script ocorrerá utilizando a linguagem de programação Python, descrita na subseção 2.6.1 (Python);
- **arquivos de entrada e saída**: O Script deve receber os arquivos de entrada (Arquivos com a extensão `.shp`) e gerar os arquivos de saída (Arquivos com a extensão `.sql`), mais informações na subseção 3.2.2;
- **envio de dados**: Após o script realizar a conversão dos dados para `.sql` o **Cliente** deve encaminhar essa consulta para o servidor;
- **recebimento de dados**: Quando um cliente estiver no mesmo ambiente do servidor, ou com acesso ao servidor, poderá realizar uma solicitação que verifica se existe atualização para ele, e a informação deve ser retornada pelo servidor.

3.2.4 Implementação do protótipo do cliente da segunda fase

A implementação dos protótipos de tela deve utilizar o software *Pencil Project*[1]. A prototipação deve obedecer as informações levantadas na análise de requisitos[2].

3.2.5 Implementação do Cliente da segunda fase

O *Cliente* da segunda fase além de utilizar Python como linguagem de programação, deve funcionar como um *plugin* para o Software QGIS, utilizando como base as funcionalidades descritas na subseção 3.2.3, porém utilizando os padrões de projeto de plugin disponíveis em (QGIS, 2011). Além de seguir as boas práticas de projeto a implementação deve seguir os protótipos de telas gerados.

3.3 Testes

A realização dos testes deve ser dividida em duas etapas a fim de atender as necessidades de cada uma das fases de implementação. Para a primeira fase os Testes de funcionamento (subseção 3.3.1) e para a segunda fase a aplicação dos Testes de usabilidade, subseção 3.3.2.

[1] É um sofware de código aberto sobre a licença *GNU Public License version 2*, disponível em: https://www.gnu.org/licenses/old-licenses/gpl-2.0.html, e pode ser baixado gratuitamente pelo link https://pencil.evolus.vn/Downloads.html

[2] Descrito em seção 3.1

3.3.1 Testes de funcionamento

A fim de garantir a integridade, transparência e praticidade dos resultados a serem gerados, se faz necessário definir o que testar e como testar. Faz-se necessário:

- utilizar arquivos e projetos do Software QGIS e realizar as suas conversões utilizando os comandos `shp2pgsql` e `pgsql2shp`;
- gerar arquivos com as diversas combinações de parâmetros dos comandos `shp2pgsql` e `pgsql2shp` disponíveis;
- encaminhar consultas para o servidor;
- recuperar informações do servidor;

3.3.2 Testes de usabilidade

A realização dos testes de usabilidade deve se basear na taxa de sucesso em cumprir tarefas, estas tarefas são as seguintes:

- atualizar configurações pela interface gráfica;
- atualizar configurações pelo arquivo de configuração;
- testar as configurações ativas;
- realizar a sincronização de dados.

As tarefas acima são as tarefas necessárias para o funcionamento do *plugin*.

4 RESULTADOS

Os resultados obtidos foram divididos em alguns grupos conforme descritos nas seções a seguir.

4.1 Análise de requisitos

Durante o processo de análise de requisitos foi possível identificar a necessidade de implementação das seguintes funcionalidades:

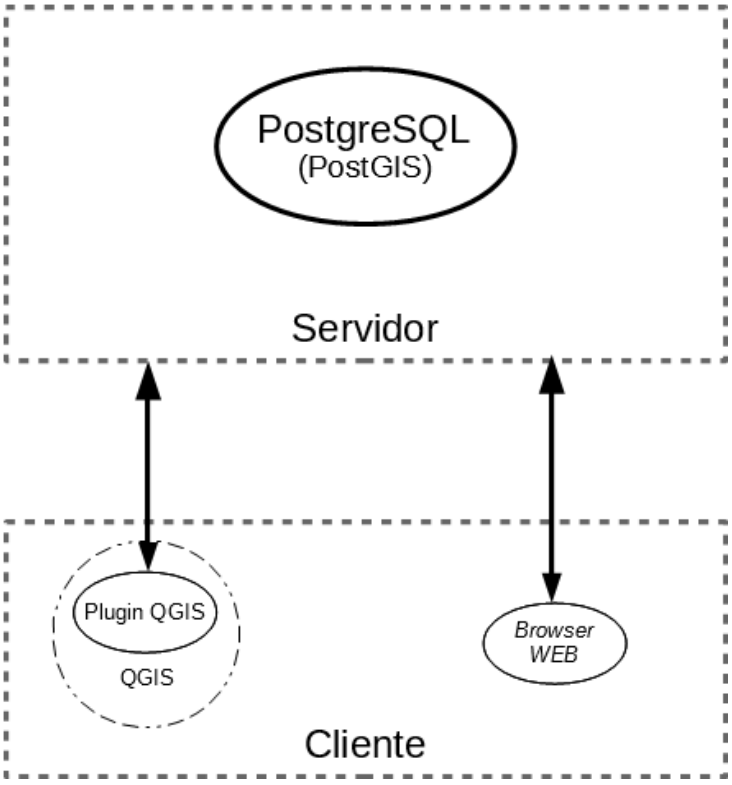
- cliente (*plugin* para o QGIS):
  - capturar os eventos ocorridos na edição de dados de um projeto no QGIS;
  - enviar as alterações para o servidor;
  - receber as atualizações do servidor;
  - o cliente não possui banco de dados.
- cliente (*browsers*):
  - visualizar mapas através de *browsers* na WEB e intranet;
  - a visualização deve possuir controle de acesso.
- servidor:
  - armazenar os mapas atualizados;
  - receber atualização de clientes;
  - enviar atualização para clientes.
  - WEB - QGIS Server:
    - permite a utilização dos padrões WMS e WTMS.

Uma simplificação desta análise pode ser visualizada na Figura 7.

Durante a revisão dos requisitos levantados, conforme descrito anteriormente nesta seção, foi possível identificar alguns desafios:

- a sincronização deverá ocorrer quando o *plugin* for carregado;
  - para realizar o envio, o arquivo deverá ser convertido para o formato `.sql`;
  - para realizar o recebimento, o arquivo deverá ser convertido para o formato `.shp`.
- o servidor deverá armazenar todos os scripts recebidos e repassar para todos clientes, se for necessário;

Figura 7: Exemplo da estrutura do servidor



Fonte: O próprio autor

- o cliente deverá manter uma data de atualização para verificar qual ou quais scripts devem ser retornados do servidor;
- o nome do banco deverá ser armazenado junto ao projeto;
- o servidor deverá receber o script e atualizar/criar as suas bases de dados;
- os usuários necessitam de níveis de acesso.

4.2 Implementação da ferramenta

A implementação desta ferramenta ocorreu em duas etapas, para poder validar o funcionamento da ferramenta e identificar outros possíveis problemas. A Primeira fase é descrita na subseção 4.2.1 e a Segunda fase descrita na subseção 4.2.2.

4.2.1 Primeira fase

O desenvolvimento da primeira fase foi realizado para validar o funcionamento da ferramenta. Onde foi desenvolvido o `Cliente`, descrito na subseção 4.2.1.3 e o `Servidor` na subseção 4.2.1.1.

4.2.1.1 Servidor da ferramenta

O servidor da ferramenta consiste em um servidor de banco de dados PostgreSQL com a extensão PostGIS, instalado sobre um sistema Linux com a distribuição Debian 9.

4.2.1.2 Geração das consultas em SQL

As consultas são geradas pelo cliente da ferramenta e convertidas para o formato local correto. Um arquivo de configuração é criado no cliente para auxiliar no gerenciamento dos arquivos que estão sendo trabalhados. Um trecho deste arquivo de configuração pode ser verificado no Script 4.1.

```
1 [LOCAL]
2 folder = F:/mapas_novo
3 last_sync = 2017-11-18T13:52:19.618000
```

Script 4.1: Trecho de arquivo de configuração gerado após atualização

4.2.1.3 Cliente

Durante a implementação da ferramenta, na parte relativa ao `Cliente`, foi possível identificar que um arquivo em formato `.shp` com aproximadamente **10 MB** quando convertido para formato `.sql` ultrapassava os **35 MB**. Desta forma buscou-se gerar arquivos menores o que por sua vez gerou os seguintes desafios:

- **como gerar arquivos menores**: utilizando um padrão utilizado em sistemas de versionamento, encaminhando somente as alterações realizadas nos arquivo. Para isso duas alternativas foram encontradas:
  - em sistemas *Unix* o comando nativo `diff` realizava a tarefa sem problemas. Porém em sistemas *Windows* não havia uma alternativa. Por isso a utilização do `diff` foi descartada;
  - o Python possui uma biblioteca, `difflib`, com o mesmo propósito de gerar a diferença entre um arquivo de entrada e outro de saída, logo, a adoção desta tecnologia foi adotada.
- **como enviar**: é necessário encaminhar o resultado da diferença para o servidor. O Script 4.2 realiza a conversão para o formato `.sql`;
- **como receber**: o `Cliente` recebe somente as atualizações que são mescladas com seu arquivo original e convertidas novamente para o formato `.shp`. A conversão é realizada através do Script 4.3;
- **processamento do arquivo recebido**: Após receber este arquivo é necessário realizar a conversão do arquivo para `.sql`, utilizando o Script 4.3. A partir deste

novo arquivo gerado e do arquivo original na máquina, é produzido um novo arquivo `.sql` utilizando o Script 4.4. Esse arquivo, menor, é encaminhado para o servidor para ser executado e novamente é realizada uma consulta no servidor para recuperar o arquivo atualizado, sobrescrevendo o arquivo do usuário.

A Figura 7 mostra a estrutura montada entre cliente e servidor, exemplificando a troca de informações entre as partes, onde o cliente encaminha os dados e o servidor recebe. As informações são atualizadas no servidor, assim o próximo cliente que tentar realizar a atualização deverá atualizar os dados locais conforme as informações armazenadas no servidor. O script em Python gera um novo arquivo local de modificação para ser encaminhado para o servidor.

O `Cliente` foi implementado em Python para facilitar a implementação de um plugin para o QGIS. A implementação do cliente roda basicamente em cima de duas funções. A primeira, descrita no Script 4.2, realiza a conversão do arquivo binário em `.shp` para o formato `.sql`. A segunda função, descrita no Script 4.3, recupera a informação armazenada no servidor e gera o arquivo `.shp`.

```
127 def create_sql(bin_folder, input_file, output_files_folder, filename, sufix = ''):
128   # nomes de arquivos de origem e atual - SQL
129   input_sql_file_name = output_files_folder + DS + filename + sufix + '.sql'
130   # print input_sql_file_name
131   # gerando arquivos para comparacao
132   args_input = bin_folder + DS + 'shp2pgsql.exe -W "latin1" -e ' + input_file + ' > ' +
        input_sql_file_name
133
134   # print args_input
135
136   try:
137     retcode = subprocess.call(args_input, shell=True)
138     if retcode < 0:
139       print >>sys.stderr, "Child was terminated by signal", -retcode
140     else:
141       print >>sys.stderr, "Child returned", retcode
142   except OSError as e:
143     print >>sys.stderr, "Execution failed:", e
144
145   return input_sql_file_name
```

Script 4.2: Conversão de .shp para .sql

```
147 def create_shp(bin_folder, output_files_folder, filename, sufix = ''):
148   # nomes de arquivos de origem e atual - SQL
149   shp_file_name = output_files_folder + DS + filename + sufix
150   # print input_sql_file_name
151   # gerando arquivos para comparacao
152   args_input = bin_folder + DS + 'pgsql2shp.exe -f "' + shp_file_name + '" -h ' +
        config_get('SERVER', 'pg_host') + \
153     ' -P ' + config_get('SERVER', 'pg_pass') + \
154     ' -u ' + config_get('SERVER', 'pg_user') + \
155     ' ' + config_get('SERVER', 'pg_database') +\
156     ' ' + filename.lower()
157
158   print args_input
```

```
159
160   try:
161     retcode = subprocess.call(args_input, shell=True)
162     if retcode < 0:
163       print >>sys.stderr, "Child was terminated by signal", -retcode
164     else:
165       print >>sys.stderr, "Child returned", retcode
166   except OSError as e:
167     print >>sys.stderr, "Execution failed:", e
168
169   config_set(filename, 'file_temp', shp_file_name + '.shp')
170
171   return shp_file_name
```

Script 4.3: Conversão de .shp para .sql

```
173 def create_diff_sql(file_a, file_b, output_file):
174   original = open(file_a, 'r')
175   updated = open(file_b, 'r')
176
177   output_file_full = output_files_folder + DS + output_file + '.sql'
178
179   # print 'arquivo a: ' + file_a
180   # print 'arquivo b: ' + file_b
181   # print 'arquivo de saida: ' + output_file
182
183   diff = difflib.ndiff(original.readlines(), updated.readlines())
184
185   delete_registers = ''.join(re.sub(r"[\s]*INSERT[\s]+INTO[\s]+\"(\w+)\" \([\"\w,]+\)\s+
      VALUES\s+\('(\d+)'[\,\'\d\w\W\s\/\-\(\)\.]+;[\s]*", r'DELETE FROM `\1` WHERE `id` =
      \2;\n', x[2:]) for x in diff if x.startswith('- '))
186   create_registers = ''.join(x[2:] for x in diff if x.startswith('+ '))
187   # print output_file_full
188   sql_export = open(output_file_full, 'wb');
189   sql_export.write(delete_registers);
190   sql_export.write(create_registers);
191   sql_export.close();
192
193   return output_file_full
```

Script 4.4: Geração do arquivo `.sql` utilizando a biblioteca `difflib`

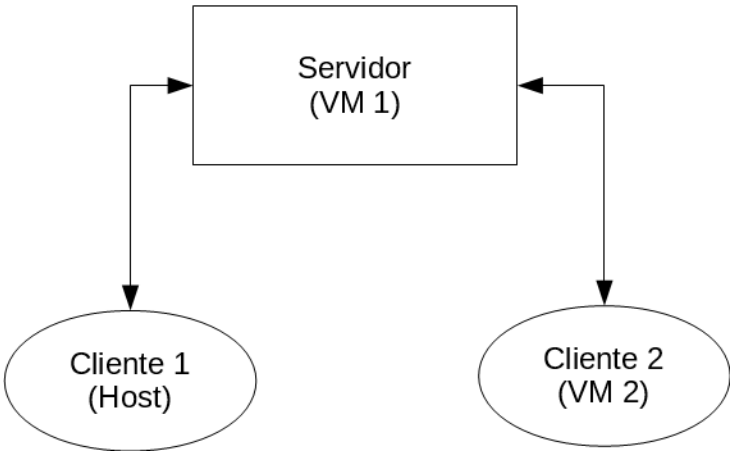
4.2.1.4 Testes

A realização dos testes ocorreu utilizando duas VMs. Assim, o envio e a recuperação de dados dos clientes com o servidor ocorreu normalmente. O seguinte cenário foi executado:

1. Cliente 1 gera um novo mapa e sincroniza com servidor;
2. Cliente 2 realiza o recebimento desse novo mapa;
3. Cliente 2 atualiza localmente o mapa;
4. Cliente 2 realiza sincronização com o servidor;

5. Cliente 1 atualiza localmente o mapa;
6. Cliente 1 realiza sincronização com o servidor;
   a) Recebe `.sql` do servidor;
   b) Gera `.sql` localmente com base no arquivo do mapa;
   c) Gera o arquivo de diferença do arquivo `.sql`;
   d) Executa o script `.sql` no servidor, somente com as modificações geradas;

Figura 8: Esquema de testes de funcionalidades



Fonte: O próprio autor

A Figura 8 exemplifica este cenário, onde o `Host` das VMs atuou como **Cliente 1**, a **VM 1** como **Servidor** e a **VM 2** como **Cliente 2**.

4.2.2 Segunda fase

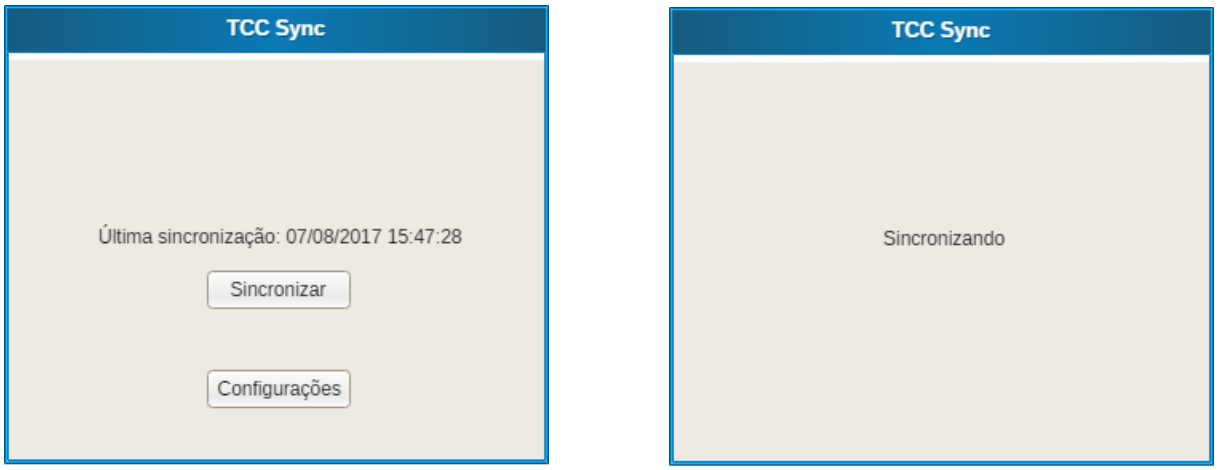
Os resultados da segunda fase, que consiste na implementação de um *plugin* para o Software QGIS, está discriminada a seguir. A subseção 4.2.2.1 descreve o protótipo gerado. A subseção 4.2.2.2 mostra a implementação final da ferramenta. Por fim, os testes de usabilidade são descritos na subseção 4.2.2.3.

4.2.2.1 Prototipação da ferramenta

Os protótipos de tela gerados para exemplificar a interação homem/máquina foram gerados utilizando o software *Pencil Project*.

A Figura 9a representa o protótipo da tela inicial do *plugin*. Ao clicar em `Sincronizar` espera-se que seja carregada a tela descrita na Figura 9b e ao final da sincronização deve retornar para a tela descrita na Figura 9a. Caso tenha ocorrido erro no processo, uma mensagem de erro deve ser exibida no local da data da última atualização.

Figura 9: Protótipos de tela: inicial e sincronização do *plugin*

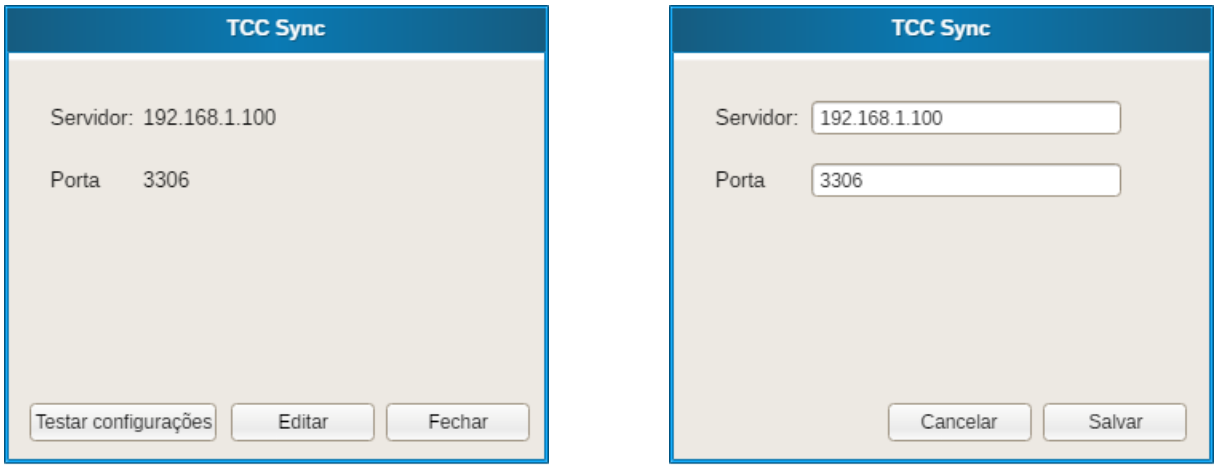


(a) Protótipo da tela inicial

(b) Protótipo da tela de sincronização

Fonte: O próprio autor

Figura 10: Protótipos da tela: configurações e editar configurações do *plugin*



(a) Protótipo da tela para exibir configurações

(b) Protótipo da tela para editar configurações

Fonte: O próprio autor

A opção `Configurações`, disponível na tela inicial, deve carregar a tela descrita na Figura 10a. Nesta tela deverão ser listadas todas as configurações que o *plugin* necessita para realizar a conexão com o servidor, permitindo que o usuário altere essas configurações de forma visual, conforme a Figura 10b. As informações de configuração devem ser carregadas e armazenadas no arquivo de configuração do *plugin*.

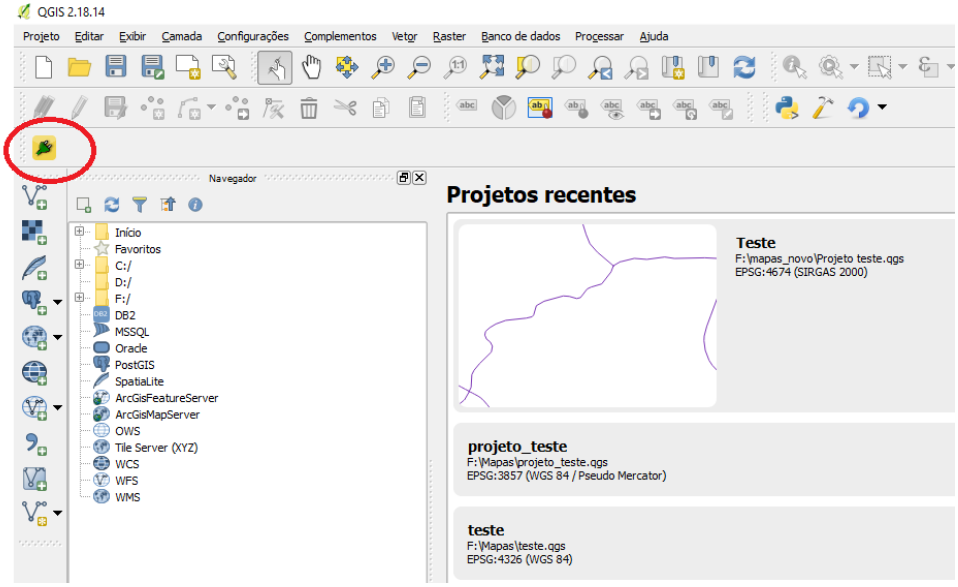
4.2.2.2 Implementação final da ferramenta

A implementação final da ferramenta, que ocorreu nesta segunda fase, foi realizada tomando como base o protótipo implementado na subseção 4.2.2.1.

A estrutura de pastas e arquivos, sugerida como boa prática na documentação oficial (QGIS, 2011), ficou na seguinte forma, tomando como base a pasta raiz do plugin:

```
/TccSyncPlugin
├── _temp_in ..... Arquivos temporários recebidos
├── _temp_out ..... Arquivos temporários enviados
├── bin ..... Arquivos binários
│   ├── libgcc_s_sjlj-1.dll
│   ├── libgeos.dll
│   ├── libgeos_c.dll
│   ├── libiconv-2.dll
│   ├── libintl-8.dll
│   ├── libpq.dll
│   ├── libproj-9.dll
│   ├── libstdc++-6.dll
│   ├── pgsql2shp.exe
│   ├── shp2pgsql.exe
│   └── ssleay32.dll
├── i18n ..... Arquivos para internacionalização
│   ├── af.ts
│   └── pt-br.ts
├── __init__.py ..... Script A.1
├── icon.png
├── metadata.txt
├── pb_tool.cfg
├── plugin_upload.py ..... Enviar pacote do plugin para o servidor do QGIS
├── pylintrc
├── qgis_sync.cfg ..... Arquivo de configuração gerado pelo plugin
├── tcc_sync.py ..... Script A.2
├── tcc_sync_dialog.py ..... Script A.3
├── tcc_sync_dialog_base.ui ..... Script B.1
├── tcc_sync_dialog_config.py ..... Script A.4
├── tcc_sync_dialog_config.ui ..... Script B.2
├── tcc_sync_dialog_config_edit.py ..... Script A.5
├── tcc_sync_dialog_config_edit.ui ..... Script B.3
├── tcc_sync_helpers.py ..... Script A.7
└── tcc_sync_helpers_sql.py ..... Script A.6
```

Figura 11: Tela inicial do QGIS com o *plugin*

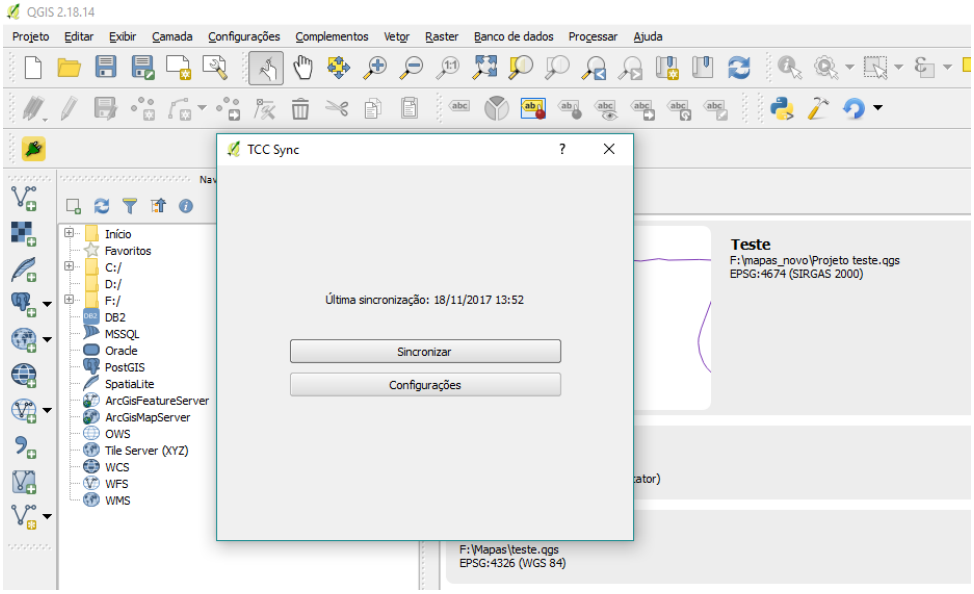


Fonte: O próprio autor

A Figura 11 mostra a tela inicial do QGIS. No canto esquerdo e destacado com um círculo encontra-se o botão do *plugin*. Ao clicar sobre o botão do *plugin* é carregado o *plugin*, conforme a Figura 12.

A tela inicial fornece duas opções, a primeira opção, `Sincronizar`, realiza o processo de sincronização descrito na subseção 3.2.5, onde o *plugin* realiza uma busca por todos arquivos com a extensão `.shp` do projeto e realiza a conversão para o formato `.sql`

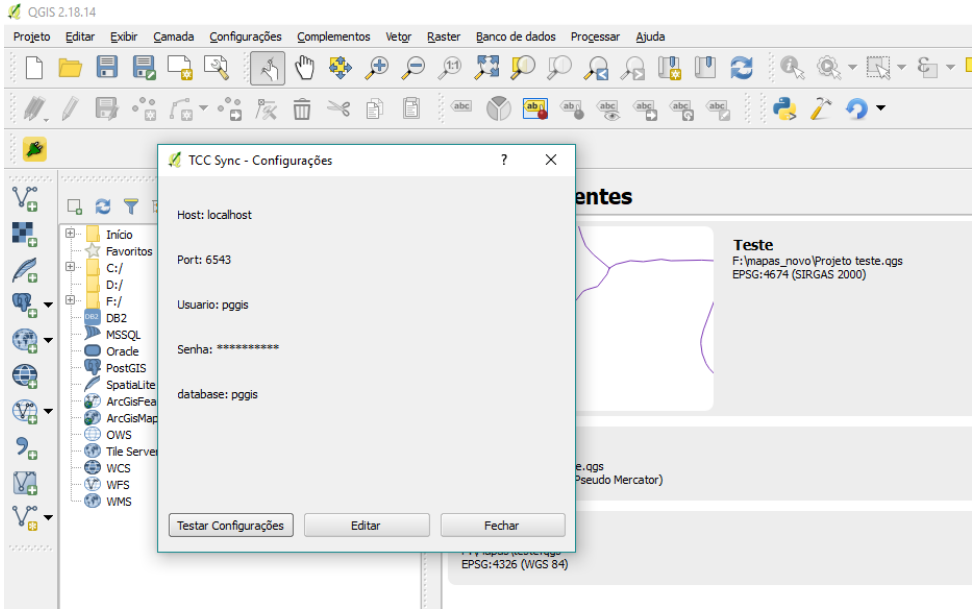
Figura 12: Tela inicial do *plugin*



Fonte: O próprio autor

armazenando estes arquivos na pasta `_temp_out`. Logo realiza a busca pelos dados do servidor. Se a informação solicitada não for localizada, o arquivo `.sql` gerado anteriormente e armazenado em `_temp_out` é encaminhado para o servidor e removido da pasta. Caso contrário, é gerado um novo arquivo `.sql` com os dados armazenados no servidor, gerando assim um outro arquivo SQL menor e somente com os dados atualizados localmente, logo após é realizada a submissão deste para o servidor.

Figura 13: Tela para visualizar as configurações do *plugin*



Fonte: O próprio autor

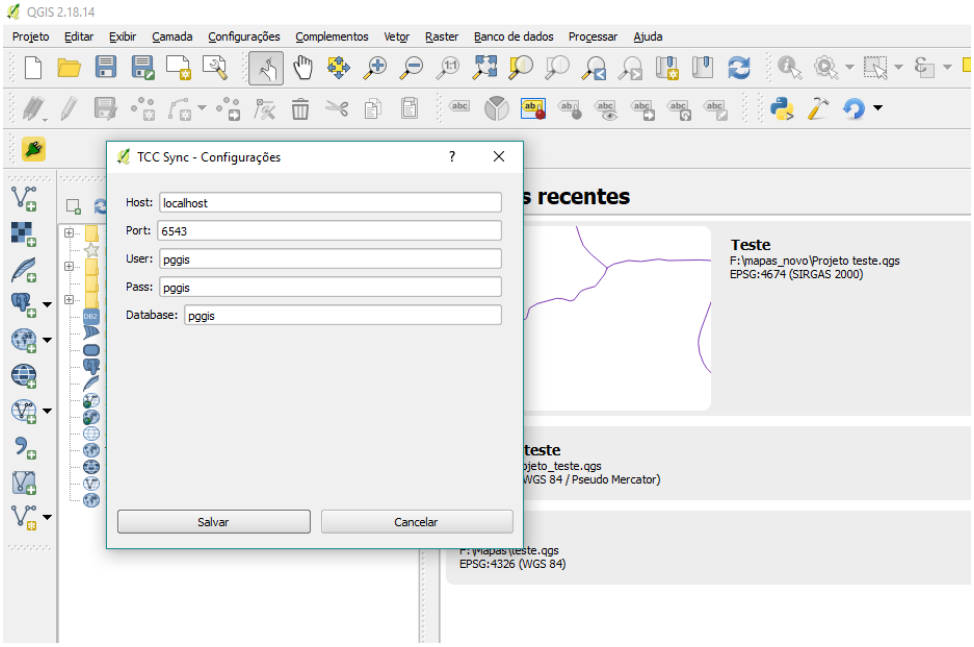
A segunda opção disponível na tela inicial, `Configurações`, carrega a tela descrita na Figura 13. Esta tela fornece acesso para realizar o teste da configuração ativa, através do botão `Testar Configurações`, que pode retornar sucesso (Figura 16) ou erro (Figura 15), também fornece acesso a tela que possibilita editar as configurações (Figura 14) através do botão `Editar`.

4.2.2.3 Testes de usabilidade

A implementação dos testes de usabilidade foi realizada seguindo o roteiro definido na subseção 3.3.2. Para apresentar o teste foi confeccionado um documento baseado no modelo disponibilizado em Usability.gov (b), e disponível em Apêndice C. Este documento foi produzido de forma que explicasse o motivo deste teste, as tarefas que deveriam ser realizadas e como seria realizada a avaliação.

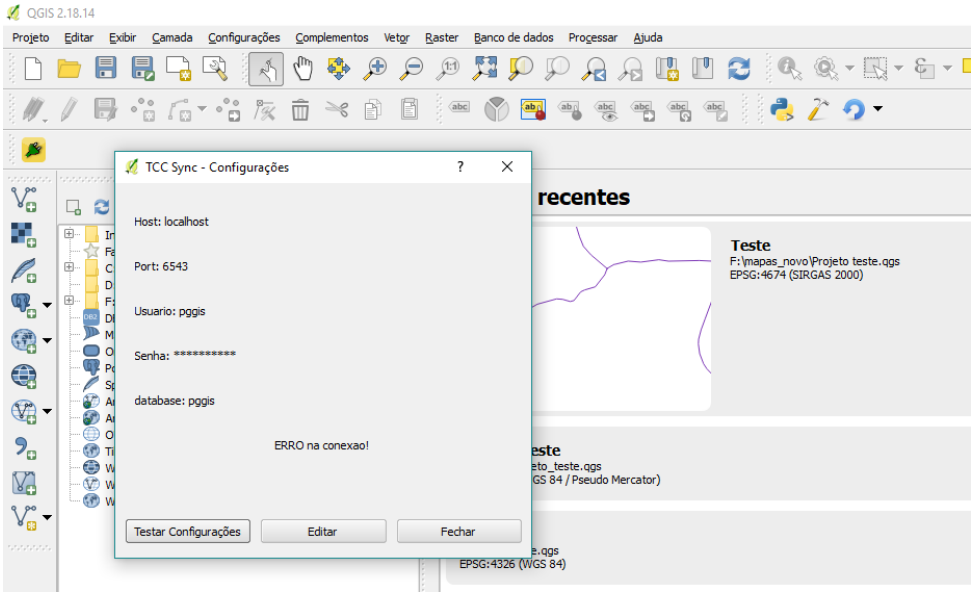
No primeiro teste de usabilidade foi identificado um **erro crítico**, o qual impedia a execução da ferramenta. Este estava relacionado a criação do arquivo de configuração do *plugin*, onde este arquivo não estava sendo criado, assim, impedindo que a ferramenta fosse inicializada. Foi realizada a correção deste erro, e pontualmente neste teste não foi

Figura 14: Tela para editar as configurações do *plugin*



Fonte: O próprio autor

Figura 15: Tela de erro no teste de configurações do *plugin*

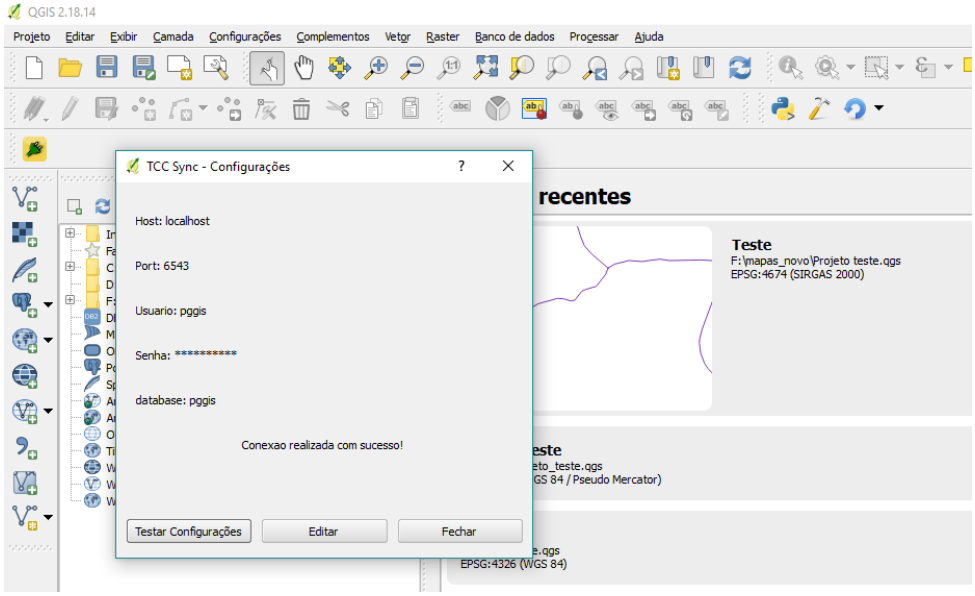


Fonte: O próprio autor

necessário reagendá-lo, sendo necessário somente redefinir o ambiente do teste e realizá-lo novamente.

Nos demais testes realizados foi percebido um comportamento não esperado da ferramenta, onde várias telas do mesmo são carregadas. Este problema está relacionado a troca de janelas, porém ainda sem solução. Ele foi classificado com severidade média,

Figura 16: Tela de sucesso no teste de configurações do *plugin*



Fonte: O próprio autor

pois ocorre somente quando são carregadas as telas de configurações e não impede o funcionamento básico do sistema.

A Tabela 5 reune todos os resultados obtidos e disponibilizados no Apêndice D.

Tabela 5: Compilado dos resultados do teste de usabilidade

Local / Usuário	**Tarefa 1**	**Tarefa 2**	**Tarefa 3**	**Tarefa 4**
Empresa pampatec / Usuário 1	FEC	NF	F	FEN
Grupo de pesquisa / Usuário 1	FEN	F	F	FEN
Grupo de pesquisa / Usuário 2	FEN	F	F	FEN
Grupo de pesquisa / Usuário 3	FEN	F	F	FEN
Grupo de pesquisa / Usuário 4	FEN	F	F	FEN

`F` = Finalizada; `NF` = Não Finalizada; `FEC` = Finalizada com Erro Crítico; `FEN` = Finalizada com Erro Normal

5 CONSIDERAÇÕES FINAIS

Realizando uma comparação relacionada a metodologia proposta, onde foram definidos os processos para o desenvolvimento da ferramenta proposta, juntamente com os resultados obtidos neste trabalho é possível definir alguns itens, como por exemplo o cumprimento tanto do objetivo geral quanto do objetivo específico, a descoberta de problemas, permitindo assim, classificá-los em ordem de severidade.

O objetivo específico, disponível na seção 1.2, foi atingido com a implementação da ferramenta proposta. Esta ferramenta tinha como objetivo a sincronização de informações entre grupos de usuários, mas possibilita que o usuário trabalhasse de forma *off-line*[1] em relação ao servidor de trabalho.

Durante a implementação da ferramenta, inúmeros problemas foram enfrentados, entre os mais graves é possível destacar a falta de detalhamento da análise de requisitos, na qual não foi possível identificar o tamanho grande dos arquivos trabalhados. Outro problema que pode ser destacado está relacionado a fase de testes, a qual além de demandar muito tempo para sua realização, apresentou alguns problemas na sua execução, o problema foi classificado como **crítico** por impedir a utilização do sistema no teste, porém foi facilmente corrigido.

5.1 Trabalhos futuros

A informação após ser transmitida para o servidor fica armazenada em um banco de dados. Esta informação pode ser trabalhada de inúmeras maneiras, mas a mais observada, principalmente na análise de requisitos está relacionada a divulgação destas informações através dos padrões WMS ou WTMS, deixando a informação, ou parte dessa informação disponível para acesso *HTTP*.

Outra alternativa, não abordada neste trabalho, seria a mineração de dados relacionados a estas informações, pois Sistemas de Informações Geográficas conseguem armazenar diversos tipos de informações, porém estas informações muitas vezes não são exploradas de maneira adequada.

[1] Definição: (1) sem conexão a um computador associado. (2) que não está conectado a um computador ou que não pode ser us. em um dado momento (diz-se de sistema, equipamento ou dispositivo).

REFERÊNCIAS

BOEHM, B. et al. Using the winwin spiral model: a case study. **Computer**, IEEE, v. 31, n. 7, p. 33–44, 1998. Citado na página 35.

CÂMARA, G. Modelos, linguagens e arquiteturas para bancos de dados geográficos. **São José dos Campos**, v. 264, 1995. Citado na página 23.

CÂMARA, G. **Anatomia de sistemas de informação geográfica**. [S.l.]: UNICAMP-Instituto de Computação, 1996. Citado na página 22.

CÂMARA, G. Representação computacional de dados geográficos. **Bancos de Dados**, 2005. Citado 2 vezes nas páginas 21 e 23.

CAMPBELL, J. E. **Geographic Information System Basics**. [S.l.]: The Saylor Foundation, 2015. Citado na página 29.

GOMES, J.; VELHO, L. Abstraction paradigms for computer graphics. **The Visual Computer**, Springer, v. 11, n. 5, p. 227–239, 1995. Citado na página 23.

GUDWIN, R. R. Linguagens de programação. **Campinas: DCA/FEEC/UNICAMP**, 1997. Citado na página 34.

LAURINI, R.; THOMPSON, D. **Fundamentals of spatial information systems**. [S.l.]: Academic press, 1992. v. 37. Citado na página 22.

NAUR, P.; RANDELL, B. Software engineering: Report on a conference sponsored by the nato science committee, garmisch, germany, 7th to 11th october 1968. In: NATO. [S.l.], 1969. Citado na página 30.

PRESSMAN, R. S. **Engenharia de software**: Uma abordagem profissional. [S.l.]: AMGH Editora, 2011. Citado 3 vezes nas páginas 30, 31 e 35.

QGIS, D. Quantum gis geographic information system. **Open Source Geospatial Foundation Project**, 2011. Citado 3 vezes nas páginas 30, 37 e 46.

RAMSEY, P. et al. Postgis manual. **Refractions Research Inc**, 2005. Citado na página 26.

RICARTE, I. L. M. Sistemas de gerˆ encia de dados para projeto auxiliado por computador. **Disserta cão apresentada como parte dos requisitos para t tulo de mestre em engenharia el etrica, Universidade Estadual de Campinas, Faculdade de Engenharia El etrica, Dep. de Engenharia da Computa cão e Automa cão Industrial**, 1987. Citado na página 22.

ROSSUM, G. van; DRAKE, F. L. **disponível livermente na internet em http://www. python. org. br/wiki**. [S.l.]: DocumentacaoPython, 2017. Citado na página 34.

SILBERSCHATZ, A. et al. **Sistema de banco de dados**. [S.l.]: Elsevier, 2006. Citado na página 25.

STEVENS, S. S. **On the theory of scales of measurement**. [S.l.]: Bobbs-Merrill, College Division, 1946. Citado 2 vezes nas páginas 23 e 24.

TRIPP, L. Ieee standards collection software engineering. **New York, NY: Institute of Electrical and Electronics Engineers**, 1994. Citado na página 30.

USABILITY.GOV. **Planning a Usability Test**. Disponível em: <https://www.usability.gov/how-to-and-tools/methods/planning-usability-testing.html>. Citado 2 vezes nas páginas 32 e 33.

USABILITY.GOV. **Reporting Usability Test Results**. Disponível em: <https://www.usability.gov/how-to-and-tools/methods/reporting-usability-test-results.html>. Citado 2 vezes nas páginas 34 e 48.

USABILITY.GOV. **Running a Usability Test**. Disponível em: <https://www.usability.gov/how-to-and-tools/methods/running-usability-tests.html>. Citado na página 33.

USABILITY.GOV. **Usability Testing**. Disponível em: <https://www.usability.gov/how-to-and-tools/methods/usability-testing.html>. Citado na página 31.

Apêndices

APÊNDICE A – SCRIPTS EM PYTHON

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 TccSync
                                 A QGIS plugin
 Sincronização com banco de dados
                              -------------------
        begin                : 2017-08-21
        copyright            : (C) 2017 by Madson Verdi Junior
        email                : madsonverdijr@gmail.com
        git sha              : $Format:%H$
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
 This script initializes the plugin, making it known to QGIS.
"""


# noinspection PyPep8Naming
def classFactory(iface):  # pylint: disable=invalid-name
    """Load TccSync class from file TccSync.

    :param iface: A QGIS interface instance.
    :type iface: QgsInterface
    """
    #
    from .tcc_sync import TccSync
    return TccSync(iface)
```

Script A.1: classFactory

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 TccSync
                                 A QGIS plugin
 Sincronização com banco de dados
                              -------------------
        begin                : 2017-08-21
        git sha              : $Format:%H$
        copyright            : (C) 2017 by Madson Verdi Junior
        email                : madsonverdijr@gmail.com
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
```

```
21  ***************************************************************************/
22 """
23 from PyQt4.QtCore import QSettings, QTranslator, qVersion, QCoreApplication
24 from PyQt4.QtGui import QAction, QIcon
25
26 # Initialize Qt resources from file resources.py
27 import resources
28
29 # Import the code for the dialog
30 from tcc_sync_dialog import TccSyncDialog
31 from tcc_sync_dialog_config import TccSyncDialogConfig
32 from tcc_sync_dialog_config_edit import TccSyncDialogConfigEdit
33
34
35 # Import Helpers
36 from tcc_sync_helpers import TccSyncHelpers
37
38 import os.path
39
40
41 class TccSync:
42     """QGIS Plugin Implementation."""
43
44     def __init__(self, iface):
45         """Constructor.
46
47         :param iface: An interface instance that will be passed to this class
48             which provides the hook by which you can manipulate the QGIS
49             application at run time.
50         :type iface: QgsInterface
51         """
52         # Save reference to the QGIS interface
53         self.iface = iface
54         # initialize plugin directory
55         self.plugin_dir = os.path.dirname(__file__)
56         # initialize locale
57         locale = QSettings().value('locale/userLocale')[0:2]
58         locale_path = os.path.join(
59             self.plugin_dir,
60             'i18n',
61             'TccSync_{}.qm'.format(locale))
62
63         if os.path.exists(locale_path):
64             self.translator = QTranslator()
65             self.translator.load(locale_path)
66
67             if qVersion() > '4.3.3':
68                 QCoreApplication.installTranslator(self.translator)
69
70
71         # Declare instance attributes
72         self.actions = []
73         self.menu = self.tr(u'&TCC Sync')
74         # TODO: We are going to let the user set this up in a future iteration
75         self.toolbar = self.iface.addToolBar(u'TccSync')
76         self.toolbar.setObjectName(u'TccSync')
77
78         # initialize helpers
79         self.helpers = TccSyncHelpers()
80
```

```
81      # noinspection PyMethodMayBeStatic
82      def tr(self, message):
83          """Get the translation for a string using Qt translation API.
84
85          We implement this ourselves since we do not inherit QObject.
86
87          :param message: String for translation.
88          :type message: str, QString
89
90          :returns: Translated version of message.
91          :rtype: QString
92          """
93          # noinspection PyTypeChecker,PyArgumentList,PyCallByClass
94          return QCoreApplication.translate('TccSync', message)
95
96
97      def add_action(
98          self,
99          icon_path,
100         text,
101         callback,
102         enabled_flag=True,
103         add_to_menu=True,
104         add_to_toolbar=True,
105         status_tip=None,
106         whats_this=None,
107         parent=None):
108         """Add a toolbar icon to the toolbar.
109
110         :param icon_path: Path to the icon for this action. Can be a resource
111             path (e.g. ':/plugins/foo/bar.png') or a normal file system path.
112         :type icon_path: str
113
114         :param text: Text that should be shown in menu items for this action.
115         :type text: str
116
117         :param callback: Function to be called when the action is triggered.
118         :type callback: function
119
120         :param enabled_flag: A flag indicating if the action should be enabled
121             by default. Defaults to True.
122         :type enabled_flag: bool
123
124         :param add_to_menu: Flag indicating whether the action should also
125             be added to the menu. Defaults to True.
126         :type add_to_menu: bool
127
128         :param add_to_toolbar: Flag indicating whether the action should also
129             be added to the toolbar. Defaults to True.
130         :type add_to_toolbar: bool
131
132         :param status_tip: Optional text to show in a popup when mouse pointer
133             hovers over the action.
134         :type status_tip: str
135
136         :param parent: Parent widget for the new action. Defaults None.
137         :type parent: QWidget
138
139         :param whats_this: Optional text to show in the status bar when the
140             mouse pointer hovers over the action.
```

```
141
142         :returns: The action that was created. Note that the action is also
143             added to self.actions list.
144         :rtype: QAction
145         """
146
147         # Create the dialog (after translation) and keep reference
148         self.dlg = TccSyncDialog()
149         self.dlg_config = TccSyncDialogConfig()
150         self.dlg_config_edit = TccSyncDialogConfigEdit()
151
152         icon = QIcon(icon_path)
153         action = QAction(icon, text, parent)
154         action.triggered.connect(callback)
155         action.setEnabled(enabled_flag)
156
157         if status_tip is not None:
158             action.setStatusTip(status_tip)
159
160         if whats_this is not None:
161             action.setWhatsThis(whats_this)
162
163         if add_to_toolbar:
164             self.toolbar.addAction(action)
165
166         if add_to_menu:
167             self.iface.addPluginToDatabaseMenu(
168                 self.menu,
169                 action)
170
171         self.actions.append(action)
172
173         return action
174
175     def initGui(self):
176         """Create the menu entries and toolbar icons inside the QGIS GUI."""
177
178         icon_path = ':/plugins/TccSync/icon.png'
179         self.add_action(
180             icon_path,
181             text=self.tr(u'TCC Sync'),
182             callback=self.run,
183             parent=self.iface.mainWindow())
184
185
186     def unload(self):
187         """Removes the plugin menu item and icon from QGIS GUI."""
188         for action in self.actions:
189             self.iface.removePluginDatabaseMenu(
190                 self.tr(u'&TCC Sync'),
191                 action)
192             self.iface.removeToolBarIcon(action)
193         # remove the toolbar
194         del self.toolbar
195
196
197     def run(self):
198         """Run method that performs all the real work"""
199
200         # show the dialog
```

```
201         self.dlg.show()
202 
203         # Buttons events
204         self.dlg.btnConfig.clicked.connect(self.run_config)
205 
206         # Run the dialog event loop
207         result = self.dlg.exec_()
208 
209 
210     def run_config(self):
211         """Evento que exibe as configuracoes"""
212 
213         # show the dialog
214         self.dlg_config.show()
215 
216         # Define labels contents
217         self.dlg_config.lblHost.setText('Host: ' + self.helpers.configuration_load_option
    ('SERVER', 'pg_host'))
218         self.dlg_config.lblPort.setText('Port: ' + self.helpers.configuration_load_option
    ('SERVER', 'pg_port'))
219         self.dlg_config.lblUser.setText('Usuario: ' + self.helpers.
    configuration_load_option('SERVER', 'pg_user'))
220         self.dlg_config.lblDatabase.setText('database: ' + self.helpers.
    configuration_load_option('SERVER', 'pg_database'))
221 
222         # Buttons events
223         self.dlg_config.btnConfigEdit.clicked.connect(self.run_config_edit)
224         self.dlg_config.btnConfigTest.clicked.connect(lambda:self.helpers.connection_test
    (self.dlg_config))
225 
226         # Run the dialog event loop
227         result = self.dlg_config.exec_()
228 
229     def run_config_edit(self):
230         """Evento que edita as configuracoes"""
231 
232         # show the dialog
233         self.dlg_config_edit.show()
234 
235         # Define labels contents
236         self.dlg_config_edit.editHost.insert(self.helpers.configuration_load_option('
    SERVER', 'pg_host'))
237         self.dlg_config_edit.editPort.insert(self.helpers.configuration_load_option('
    SERVER', 'pg_port'))
238         self.dlg_config_edit.editPass.insert(self.helpers.configuration_load_option('
    SERVER', 'pg_pass'))
239         self.dlg_config_edit.editUser.insert(self.helpers.configuration_load_option('
    SERVER', 'pg_user'))
240         self.dlg_config_edit.editDatabase.insert(self.helpers.configuration_load_option('
    SERVER', 'pg_database'))
241 
242         # Buttons events
243         # self.dlg_config_edit.btnSave.connect(lambda:self.helpers.
    configuration_save_options(self.dlg_config_edit, 'SERVER', []))
244 
245         # Run the dialog event loop
246         result = self.dlg_config_edit.exec_()
```

Script A.2: TccSync

```
# -*- coding: utf-8 -*-
"""
/***************************************************************************
 TccSyncDialog
                                 A QGIS plugin
 Sincronização com banco de dados
                             -------------------
        begin                : 2017-08-21
        git sha              : $Format:%H$
        copyright            : (C) 2017 by Madson Verdi Junior
        email                : madsonverdijr@gmail.com
 ***************************************************************************/

/***************************************************************************
 *                                                                         *
 *   This program is free software; you can redistribute it and/or modify  *
 *   it under the terms of the GNU General Public License as published by  *
 *   the Free Software Foundation; either version 2 of the License, or     *
 *   (at your option) any later version.                                   *
 *                                                                         *
 ***************************************************************************/
"""

import os

from PyQt4 import QtGui, uic
from datetime import datetime

# Import Helpers
from tcc_sync_helpers import TccSyncHelpers
from tcc_sync_helpers_sql import TccSyncHelpersSql

FORM_CLASS, _ = uic.loadUiType(os.path.join(
    os.path.dirname(__file__), 'tcc_sync_dialog_base.ui'))


class TccSyncDialog(QtGui.QDialog, FORM_CLASS):
    def __init__(self, parent=None):
        # Constructor.
        super(TccSyncDialog, self).__init__(parent)
        # Set up the user interface from Designer.
        # After setupUI you can access any designer object by doing
        # self.<objectname>, and you can use autoconnect slots - see
        # http://qt-project.org/doc/qt-4.8/designer-using-a-ui-file.html
        # #widgets-and-dialogs-with-auto-connect
        self.setupUi(self)
        self.btnConfig.clicked.connect(self.btn_accept)
        self.btnSync.clicked.connect(self.btn_sync)

        # initialize helpers
        self.helpers = TccSyncHelpers()
        self.helpersSql = TccSyncHelpersSql()

        self.check_last_sync()


    def btn_accept(self):
        """Close function for btn"""
        self.accept()

```

```
61      def btn_sync(self):
62          options = {}
63
64          dt = datetime.now()
65          options['last_sync'] = dt.isoformat()
66          options['folder'] = self.helpers.get_project_folder()
67
68          config = {}
69          config['pg_host'] = self.helpers.configuration_load_option('SERVER', 'pg_host')
70          config['pg_port'] = self.helpers.configuration_load_option('SERVER', 'pg_port')
71          config['pg_database'] = self.helpers.configuration_load_option('SERVER', '
    pg_database')
72          config['pg_user'] = self.helpers.configuration_load_option('SERVER', 'pg_user')
73          config['pg_pass'] = self.helpers.configuration_load_option('SERVER', 'pg_pass')
74
75          if self.helpers.connection_test(self):
76              localFolder = self.helpers.configuration_load_option(self.helpers.section, '
    folder')
77              for file, shape in self.helpers.search_files(localFolder):
78                  if not file.endswith('.shp'):
79                      temp = file
80                      file = shape
81                      shape = temp
82
83
84                  # Import from database
85                  shp_in = self.helpersSql.create_shp(self.helpers.bin_dir, self.helpers.
    temp_in_dir, shape, config)
86
87                  # For use in diff file
88                  sql_out = self.helpersSql.create_sql(self.helpers.bin_dir, file, self.
    helpers.temp_out_dir, shape, '_original')
89                  if shp_in:
90                      print 'shp_in'
91                      sql_in = self.helpersSql.create_sql(self.helpers.bin_dir, shp_in,
    self.helpers.temp_in_dir, shape)
92                      # Generate Diff for send-back to server
93                      sql_diff = self.helpersSql.create_diff_sql(self.helpers.temp_out_dir,
     sql_out, sql_in, file)
94                      self.helpersSql.send_sql_to_server(sql_diff, config)
95                  else:
96                      self.helpersSql.send_sql_to_server(sql_out, config)
97
98
99
100             # set QT texts
101             self.lblTestResult.setText('')
102             self.txtSyncStatus.setText(u'Última sincronização: ' + dt.strftime('%d/%m/%Y
    %H:%M'))
103
104             # set config update
105             self.helpers.configuration_save_options(self, self.helpers.section, options)
106         else:
107             self.txtSyncStatus.setText(u'Erro na conexão!')
108
109     def check_last_sync(self):
110
111         lastSync = self.helpers.configuration_load_option('LOCAL', 'last_sync')
112
113         if lastSync == ('No section error' or 'No option error'):
```

```
114             lastSync = 'Pressione "'+self.btnSync.text()+'"'
115         else:
116             dt = datetime.strptime(lastSync, "%Y-%m-%dT%H:%M:%S.%f")
117             lastSync = u'Última sincronização: ' + dt.strftime('%d/%m/%Y %H:%M')
118 
119         self.txtSyncStatus.setText(lastSync)
```

Script A.3: TccSyncDialog

```
1  import os
2  
3  from PyQt4 import QtGui, uic
4  
5  FORM_CLASS, _ = uic.loadUiType(os.path.join(
6      os.path.dirname(__file__), 'tcc_sync_dialog_config.ui'))
7  
8  
9  class TccSyncDialogConfig(QtGui.QDialog, FORM_CLASS):
10     def __init__(self, parent=None):
11         """Constructor."""
12         super(TccSyncDialogConfig, self).__init__(parent)
13         # Set up the user interface from Designer.
14         # After setupUI you can access any designer object by doing
15         # self.<objectname>, and you can use autoconnect slots - see
16         # http://qt-project.org/doc/qt-4.8/designer-using-a-ui-file.html
17         # #widgets-and-dialogs-with-auto-connect
18         self.setupUi(self)
19         self.btnConfigClose.clicked.connect(self.btn_close)
20         self.btnConfigEdit.clicked.connect(self.btn_close)
21 
22     def btn_close(self):
23         """Close function for btn"""
24         self.lblTestResult.setText('')
25         self.accept()
```

Script A.4: TccSyncDialogConfig

```
1  import os
2  import pdb
3  
4  from PyQt4 import QtGui, uic
5  
6  # Import Helpers
7  from tcc_sync_helpers import TccSyncHelpers
8  
9  FORM_CLASS, _ = uic.loadUiType(os.path.join(
10     os.path.dirname(__file__), 'tcc_sync_dialog_config_edit.ui'))
11 
12 
13 class TccSyncDialogConfigEdit(QtGui.QDialog, FORM_CLASS):
14     def __init__(self, parent=None):
15         """Constructor."""
16         super(TccSyncDialogConfigEdit, self).__init__(parent)
17         # Set up the user interface from Designer.
18         # After setupUI you can access any designer object by doing
19         # self.<objectname>, and you can use autoconnect slots - see
20         # http://qt-project.org/doc/qt-4.8/designer-using-a-ui-file.html
21         # #widgets-and-dialogs-with-auto-connect
22         self.setupUi(self)
```

```
23          self.configs = []
24          self.btnCancel.clicked.connect(self.btn_cancel)
25          self.btnSave.clicked.connect(self.btn_save)
26
27          # initialize helpers
28          self.helpers = TccSyncHelpers()
29
30      def btn_save(self):
31          # self.configuration_save_options()
32          section = 'SERVER'
33          options = {}
34          options['pg_host'] = self.editHost.text()
35          options['pg_port'] = self.editPort.text()
36          options['pg_database'] = self.editPass.text()
37          options['pg_user'] = self.editUser.text()
38          options['pg_pass'] = self.editDatabase.text()
39          self.editHost.clear()
40          self.editPort.clear()
41          self.editPass.clear()
42          self.editUser.clear()
43          self.editDatabase.clear()
44
45
46
47          self.helpers.configuration_save_options(self, section, options)
48
49          self.accept()
50
51
52
53      def btn_cancel(self):
54          """Close function for btn"""
55          self.editHost.clear()
56          self.editPort.clear()
57          self.editPass.clear()
58          self.editUser.clear()
59          self.editDatabase.clear()
60          self.accept()
```

Script A.5: TccSyncDialogConfigEdit

```
1  import os
2  import subprocess
3  import sys
4  import psycopg2
5
6  # Directory Separator (DS)
7  DS = '/'
8
9
10 class TccSyncHelpersSql:
11   """SQL Helpers for QGIS Plugin"""
12
13   def __init__(self):
14     pass
15
16   def send_sql_to_server(self, file, configs = {}):
17     try:
18       connectionString =  "dbname='"+configs['pg_database']+\
19                 "' user='"+configs['pg_user']+\
```

```
20                 "' host='"+configs['pg_host']+\
21                 "' password='"+configs['pg_pass']+\
22                 "' port='"+configs['pg_port']+"'"
23       print(connectionString, file)
24       conn = psycopg2.connect(connectionString)
25       cursor = conn.cursor()
26       sql_file = open(file, 'r')
27       print(sql_file)
28       cursor.execute(sql_file.read())
29       conn.commit()
30       cursor.close()
31       conn.close()
32       return True
33     except:
34       print "Unexpected error:",
35       print(sys.exc_info())
36
37     return False
38
39   def create_shp(self, bin_folder, output_files_folder, filename, config = {}, sufix = ''
     ):
40     # nomes de arquivos de origem e atual - SQL
41     shp_file_name = output_files_folder + DS + filename + sufix
42     # print input_sql_file_name
43     # gerando arquivos para comparacao
44     args_input = bin_folder + DS + 'pgsql2shp.exe -f "' + shp_file_name + '" -h ' +
     config['pg_host'] + \
45       ' -P ' + config['pg_pass'] + \
46       ' -u ' + config['pg_user'] + \
47       ' -p ' + config['pg_port'] + \
48       ' ' + config['pg_database'] +\
49       ' "' + filename.lower() + '"'
50
51     print args_input
52
53     try:
54       retcode = subprocess.call(args_input, shell=True)
55       if retcode < 0:
56         print >>sys.stderr, "Child was terminated by signal", -retcode
57       else:
58         print >>sys.stderr, "Child returned", retcode
59         return False
60     except OSError as e:
61       print >>sys.stderr, "Execution failed:", e
62       return False
63
64     # config_set(filename, 'file_temp', shp_file_name + '.shp')
65     return shp_file_name
66
67   def create_sql(self, bin_folder, input_file, output_files_folder, filename, sufix = '')
     :
68     # nomes de arquivos de origem e atual - SQL
69     input_sql_file_name = output_files_folder + DS + filename.replace('.shp', '') + sufix
      + '.sql'
70     print input_sql_file_name
71     # gerando arquivos para comparacao
72     args_input = bin_folder + DS + 'shp2pgsql.exe -W "latin1" -e "' + input_file +  '"  >
      "' + input_sql_file_name + '"'
73
74     print args_input
```

```
75
76      try:
77        retcode = subprocess.call(args_input, shell=True)
78        if retcode < 0:
79          print >>sys.stderr, "Child was terminated by signal", -retcode
80        else:
81          print >>sys.stderr, "Child returned", retcode
82      except OSError as e:
83        print >>sys.stderr, "Execution failed:", e
84
85      return input_sql_file_name
86
87    def create_diff_sql(self, output_files_folder, file_a, file_b, output_file):
88      original = open(file_a, 'r')
89      updated = open(file_b, 'r')
90
91      output_file_full = output_files_folder + DS + output_file + '.sql'
92
93      diff = difflib.ndiff(original.readlines(), updated.readlines())
94
95      delete_registers = ''.join(re.sub(r"[\s]*INSERT[\s]+INTO[\s]+\"(\w+)\" \([\"\w,]+\)\s
        +VALUES\s+\('(\d+)'[\,\'\d\w\W\s\/\-\(\)\.]+;[\s]*", r'DELETE FROM `\1` WHERE `id` =
        \2;\n', x[2:]) for x in diff if x.startswith('- '))
96      create_registers = ''.join(x[2:] for x in diff if x.startswith('+ '))
97
98      sql_export = open(output_file_full, 'wb');
99      sql_export.write(delete_registers);
100     sql_export.write(create_registers);
101     sql_export.close();
102
103     return output_file_full
```

Script A.6: TccSyncHelpersSql

```
1  import ConfigParser
2  import datetime
3  import difflib
4  import fnmatch
5  import httplib
6  import json
7  import os
8  import re
9  import subprocess
10 import sys
11 import urllib
12 import psycopg2
13
14 from qgis.core import QgsProject
15
16 # Directory Separator (DS)
17 DS = '/'
18
19
20 class TccSyncHelpers:
21   """Helpers for QGIS Plugin"""
22
23   def __init__(self):
24
25     self.plugin_dir = os.path.dirname(__file__).replace('\\', '/')
26     self.temp_out_dir = self.plugin_dir + DS + '_temp_out'
```

```
27      self.temp_in_dir = self.plugin_dir + DS + '_temp_in'
28      self.bin_dir = self.plugin_dir + DS + 'bin'
29      self.config = ConfigParser.RawConfigParser()
30      self.config_file_name = 'qgis_sync.cfg'
31      self.config.read(self.plugin_dir + DS + self.config_file_name)
32
33      # No Project
34      self.project = None
35
36      # Default Section
37      self.section = 'LOCAL'
38
39      if not os.path.exists(self.temp_out_dir):
40        os.makedirs(self.temp_out_dir)
41
42      if not os.path.exists(self.temp_in_dir):
43        os.makedirs(self.temp_in_dir)
44
45      # print self.plugin_dir
46      # print self.temp_out_dir
47      # print self.bin_dir
48
49      if not self.configuration_file_check():
50        # need to create a config file
51        print 'arquivo de configuracao precisa ser criado'
52        self.configuration_file_create()
53        pass
54      pass
55
56    # Verify if config file was created
57    def configuration_file_check(self):
58      file = self.plugin_dir + DS + self.config_file_name
59      check = os.path.isfile(file) and os.access(file, os.R_OK)
60      return check
61
62    # Create an initial config file with default content
63    def configuration_file_create(self):
64      cfgfile = open(self.plugin_dir + DS + self.config_file_name,'wb')
65
66      # add the settings to the structure of the file, and lets write it out...
67      self.config.add_section('SERVER')
68      self.config.set('SERVER', 'pg_host', 'localhost')
69      self.config.set('SERVER', 'pg_port', '5432')
70      self.config.set('SERVER', 'pg_database', 'tcc_sync')
71      self.config.set('SERVER', 'pg_user', 'postgres')
72      self.config.set('SERVER', 'pg_pass', '123456')
73
74      with cfgfile as cfg:
75        self.config.write(cfg)
76
77      cfgfile.close()
78
79      # print self.config.sections()
80
81    def configuration_load_option(self, section, option):
82      try:
83        return self.config.get(section, option)
84      except ConfigParser.NoSectionError as e:
85        return 'No section error'
86      except ConfigParser.NoOptionError as e:
```

```
87        return 'No option error'
88      except:
89        return 'Unknown error'
90
91    def configuration_save_options(self, dlg, section, options):
92      """Store configurations (options) in .cfg file"""
93      cfgfile = open(self.plugin_dir + DS + self.config_file_name,'wb')
94
95      # add section if not exists
96      if not self.config.has_section(section):
97        self.config.add_section(section)
98
99      # add options
100     for option in options:
101       self.config.set(section, option, options[option])
102
103     # print self.config
104
105     with cfgfile as cfg:
106       self.config.write(cfg)
107
108     cfgfile.close()
109
110   # Testa a conexao com o servidor de banco de dados
111   def connection_test(self, dlg):
112     try:
113       connectionString = "dbname='"+self.config.get('SERVER', 'pg_database')+"' user='"+
      self.config.get('SERVER', 'pg_user')+"' host='"+self.config.get('SERVER', 'pg_host')+
      "' password='"+self.config.get('SERVER', 'pg_pass')+"' port='"+self.config.get('
      SERVER', 'pg_port')+"'"
114       # print connectionString
115       conn = psycopg2.connect(connectionString)
116       dlg.lblTestResult.setText(u'Conexao realizada com sucesso!')
117
118       return True
119     except ConfigParser.NoSectionError as e:
120       dlg.lblTestResult.setText("No section error")
121     except:
122       dlg.lblTestResult.setText(u'ERRO na conexao!')
123
124     return False
125
126   # Altera entre DialogA e DialogB
127   def dialog_change(self, dlg_a, dlg_b):
128     dlg_a.setVisible(False)
129     dlg_a.setResult(1)
130
131     dlg_b.setVisible(True)
132
133     return dlg_a.result()
134
135   def dialog_close(self, dlg):
136     pass
137
138
139   def get_project_folder(self):
140
141     if self.project is None:
142       self.project = QgsProject.instance()
143
```

```
144       if self.project is not None and self.project.homePath() is not None:
145         self.section = 'LOCAL'
146       else :
147         self.section = self.project.homePath()
148
149
150     return self.project.homePath()
151
152   def search_files(self, base_folder, extension = 'shp'):
153     matches = []
154     for root, dirnames, filenames in os.walk(base_folder + DS):
155       for filename in fnmatch.filter(filenames, '*.' + extension):
156         new_filename = filename.replace('.' + extension, '')
157         date = os.path.getmtime(os.path.join(root, filename))
158         local_aux = new_filename
159         matches.append({os.path.join(root, filename).replace('\\', '/'), new_filename})
160         # create_config(new_filename, [{'file_original': os.path.join(root, filename), '
    local': date, 'server': ''}], base_folder)
161     return matches
```

Script A.7: TccSyncHelpers

APÊNDICE B – SCRIPTS PARA INTERFACE

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>TccSyncDialogBase</class>
 <widget class="QDialog" name="TccSyncDialogBase">
  <property name="windowModality">
   <enum>Qt::NonModal</enum>
  </property>
  <property name="enabled">
   <bool>true</bool>
  </property>
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>400</width>
    <height>360</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>TCC Sync</string>
  </property>
  <property name="sizeGripEnabled">
   <bool>false</bool>
  </property>
  <widget class="QWidget" name="verticalLayoutWidget">
   <property name="geometry">
    <rect>
     <x>70</x>
     <y>120</y>
     <width>261</width>
     <height>102</height>
    </rect>
   </property>
   <layout class="QVBoxLayout" name="verticalLayout">
    <property name="sizeConstraint">
     <enum>QLayout::SetMinimumSize</enum>
    </property>
    <item>
     <widget class="QLabel" name="txtSyncStatus">
      <property name="text">
       <string>Ultima sincronização: 10/09/2017 13:57</string>
      </property>
      <property name="alignment">
       <set>Qt::AlignCenter</set>
      </property>
     </widget>
    </item>
    <item>
     <widget class="QLabel" name="lblTestResult">
      <property name="text">
       <string/>
      </property>
      <property name="alignment">
       <set>Qt::AlignCenter</set>
      </property>
     </widget>
    </item>
    <item>
```

```
59       <widget class="QPushButton" name="btnSync">
60        <property name="enabled">
61         <bool>true</bool>
62        </property>
63        <property name="text">
64         <string>Sincronizar</string>
65        </property>
66       </widget>
67      </item>
68      <item>
69       <widget class="QPushButton" name="btnConfig">
70        <property name="toolTip">
71         <string>&lt;html&gt;&lt;head/&gt;&lt;body&gt;&lt;p&gt;Configurações para
      gerenciamento do plugin.&lt;/p&gt;&lt;/body&gt;&lt;/html&gt;</string>
72        </property>
73        <property name="text">
74         <string>Configurações</string>
75        </property>
76       </widget>
77      </item>
78     </layout>
79    </widget>
80   </widget>
81   <resources/>
82   <connections/>
83  </ui>
```

Script B.1: Interface para TccSyncDialog

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ui version="4.0">
3   <class>TccSyncDialogConfig</class>
4   <widget class="QDialog" name="TccSyncDialogConfig">
5    <property name="windowModality">
6     <enum>Qt::NonModal</enum>
7    </property>
8    <property name="enabled">
9     <bool>true</bool>
10   </property>
11   <property name="geometry">
12    <rect>
13     <x>0</x>
14     <y>0</y>
15     <width>400</width>
16     <height>360</height>
17    </rect>
18   </property>
19   <property name="windowTitle">
20    <string>TCC Sync - Configurações</string>
21   </property>
22   <property name="sizeGripEnabled">
23    <bool>false</bool>
24   </property>
25   <widget class="QWidget" name="horizontalLayoutWidget">
26    <property name="geometry">
27     <rect>
28      <x>0</x>
29      <y>309</y>
30      <width>401</width>
31      <height>51</height>
```

```
32      </rect>
33     </property>
34     <layout class="QHBoxLayout" name="horizontalLayout">
35      <property name="leftMargin">
36       <number>10</number>
37      </property>
38      <property name="rightMargin">
39       <number>10</number>
40      </property>
41      <item>
42       <widget class="QPushButton" name="btnConfigTest">
43        <property name="text">
44         <string>Testar Configurações</string>
45        </property>
46       </widget>
47      </item>
48      <item>
49       <widget class="QPushButton" name="btnConfigEdit">
50        <property name="text">
51         <string>Editar</string>
52        </property>
53       </widget>
54      </item>
55      <item>
56       <widget class="QPushButton" name="btnConfigClose">
57        <property name="text">
58         <string>Fechar</string>
59        </property>
60       </widget>
61      </item>
62     </layout>
63    </widget>
64    <widget class="QWidget" name="verticalLayoutWidget">
65     <property name="geometry">
66      <rect>
67       <x>-1</x>
68       <y>9</y>
69       <width>401</width>
70       <height>271</height>
71      </rect>
72     </property>
73     <layout class="QVBoxLayout" name="verticalLayout">
74      <property name="leftMargin">
75       <number>20</number>
76      </property>
77      <property name="rightMargin">
78       <number>20</number>
79      </property>
80      <item>
81       <widget class="QLabel" name="lblHost">
82        <property name="text">
83         <string>Host: localhost</string>
84        </property>
85       </widget>
86      </item>
87      <item>
88       <widget class="QLabel" name="lblPort">
89        <property name="text">
90         <string>port: 8000</string>
91        </property>
```

```
92       </widget>
93      </item>
94      <item>
95       <widget class="QLabel" name="lblUser">
96        <property name="text">
97         <string>Usuário: pg_test</string>
98        </property>
99       </widget>
100     </item>
101     <item>
102      <widget class="QLabel" name="lblPassword">
103       <property name="text">
104        <string>Senha: **********</string>
105       </property>
106      </widget>
107     </item>
108     <item>
109      <widget class="QLabel" name="lblDatabase">
110       <property name="text">
111        <string>Database</string>
112       </property>
113      </widget>
114     </item>
115     <item>
116      <widget class="QLabel" name="lblTestResult">
117       <property name="text">
118        <string/>
119       </property>
120       <property name="alignment">
121        <set>Qt::AlignCenter</set>
122       </property>
123      </widget>
124     </item>
125    </layout>
126   </widget>
127  </widget>
128  <resources/>
129  <connections/>
130 </ui>
```

Script B.2: Interface para TccSyncDialogConfig

```
1  <?xml version="1.0" encoding="UTF-8"?>
2  <ui version="4.0">
3   <class>TccSyncDialogConfigEdit</class>
4   <widget class="QDialog" name="TccSyncDialogConfigEdit">
5    <property name="windowModality">
6     <enum>Qt::NonModal</enum>
7    </property>
8    <property name="enabled">
9     <bool>true</bool>
10   </property>
11   <property name="geometry">
12    <rect>
13     <x>0</x>
14     <y>0</y>
15     <width>400</width>
16     <height>360</height>
17    </rect>
18   </property>
```

```
19  <property name="windowTitle">
20   <string>TCC Sync - Configurações</string>
21  </property>
22  <property name="sizeGripEnabled">
23   <bool>false</bool>
24  </property>
25  <widget class="QWidget" name="horizontalLayoutWidget">
26   <property name="geometry">
27    <rect>
28     <x>0</x>
29     <y>309</y>
30     <width>401</width>
31     <height>51</height>
32    </rect>
33   </property>
34   <layout class="QHBoxLayout" name="horizontalLayout">
35    <property name="leftMargin">
36     <number>10</number>
37    </property>
38    <property name="rightMargin">
39     <number>10</number>
40    </property>
41    <item>
42     <widget class="QPushButton" name="btnSave">
43      <property name="text">
44       <string>Salvar</string>
45      </property>
46     </widget>
47    </item>
48    <item>
49     <widget class="QPushButton" name="btnCancel">
50      <property name="text">
51       <string>Cancelar</string>
52      </property>
53     </widget>
54    </item>
55   </layout>
56  </widget>
57  <widget class="QWidget" name="verticalLayoutWidget">
58   <property name="geometry">
59    <rect>
60     <x>-1</x>
61     <y>9</y>
62     <width>401</width>
63     <height>271</height>
64    </rect>
65   </property>
66   <layout class="QVBoxLayout" name="verticalLayout">
67    <property name="leftMargin">
68     <number>20</number>
69    </property>
70    <property name="rightMargin">
71     <number>20</number>
72    </property>
73    <item>
74     <layout class="QHBoxLayout" name="horizontalLayoutHost">
75      <item>
76       <widget class="QLabel" name="lblHost">
77        <property name="text">
78         <string>Host:</string>
```

```
79          </property>
80         </widget>
81        </item>
82        <item>
83         <widget class="QLineEdit" name="editHost"/>
84        </item>
85       </layout>
86      </item>
87      <item>
88       <layout class="QHBoxLayout" name="horizontalLayoutPort">
89        <item>
90         <widget class="QLabel" name="lblPort">
91          <property name="text">
92           <string>Port:</string>
93          </property>
94         </widget>
95        </item>
96        <item>
97         <widget class="QLineEdit" name="editPort"/>
98        </item>
99       </layout>
100     </item>
101     <item>
102      <layout class="QHBoxLayout" name="horizontalLayoutUser">
103       <item>
104        <widget class="QLabel" name="lblUser">
105         <property name="text">
106          <string>User:</string>
107         </property>
108        </widget>
109       </item>
110       <item>
111        <widget class="QLineEdit" name="editUser"/>
112       </item>
113      </layout>
114     </item>
115     <item>
116      <layout class="QHBoxLayout" name="horizontalLayoutPass">
117       <item>
118        <widget class="QLabel" name="lblPass">
119         <property name="text">
120          <string>Pass:</string>
121         </property>
122        </widget>
123       </item>
124       <item>
125        <widget class="QLineEdit" name="editPass"/>
126       </item>
127      </layout>
128     </item>
129     <item>
130      <layout class="QHBoxLayout" name="horizontalLayoutDatabase">
131       <item>
132        <widget class="QLabel" name="lblDatabase">
133         <property name="text">
134          <string>Database:</string>
135         </property>
136        </widget>
137       </item>
138       <item>
```

```
139         <widget class="QLineEdit" name="editDatabase"/>
140        </item>
141       </layout>
142      </item>
143      <item>
144       <widget class="QLabel" name="lblTestResult">
145        <property name="text">
146         <string/>
147        </property>
148        <property name="alignment">
149         <set>Qt::AlignCenter</set>
150        </property>
151       </widget>
152      </item>
153     </layout>
154    </widget>
155   </widget>
156   <resources/>
157   <connections/>
158 </ui>
```

Script B.3: Interface para TccSyncDialogConfigEdit

APÊNDICE C – PLANO DO TESTE DE USABILIDADE

Plano de teste de usabilidade

Ferramenta para sincronização de conteúdos produzidos pelo Software QGIS

Madson Verdi Junior

Novembro de 2017

1 Visão geral

Este plano foi desenvolvido com o objetivo de explanar e apresentar como será realizado o teste de usabilidade para o projeto **Ferramenta para sincronização de conteúdos produzidos pelo Software QGIS**.

2 Metodologia

Este teste tem como objetivo realizar o teste de usabilidade com todos envolvidos que utilizam o Software QGIS no meio de trabalho, seja este acadêmico ou da iniciativa privada. Onde serão analisados o cumprimento de tarefas e as sugestões de melhorias para o projeto.

2.1 Participantes

Usuários que utilizam o software QGIS como meio de trabalho.

2.2 Treinamento

Será realizada uma breve apresentação da Ferramenta (*plugin*) a ser utilizada neste teste.

2.3 Procedimento

Como laboratório para execução deste teste será utilizado o computador de trabalho do usuário e como servidor desta aplicação será fornecido um servidor temporário para armazenar estas informações. Para cada tarefa será tomado de observações, o observador não pode interferir no teste a menos que o usuário não consiga prosseguir, neste cenário o observador deve tomar nota do motivo e classificar a tarefa como não concluída.

3 Tarefas

Tarefas a serem executadas:

- atualizar configurações pela interface gráfica;
- atualizar configurações pelo arquivo de configuração;
- testar as configurações ativas;
- realizar a sincronização de dados.

4 Métricas

Será avaliado o cumprimento dessas tarefas, no caso de erros estes serão avaliados conforme o nível de severidade:

- Crítico: Impossível de prosseguir com o cumprimento da tarefa.
- Normal: Erro grave, porém ainda é possível seguir com o cumprimento da tarefa
- Baixo: Gera um erro e o usuário consegue facilmente recuperar o andamento da tarefa

5 Objetivos

Os objetivos deste teste são simples, bastando finalizar as tarefas propostas. Será avaliado se o usuário conseguir cumprir as tarefas propostas sem erros, ou somente com erros de severidade baixa.

6 Relatórios

Ao final da realização dos testes será realizado um relatório com o compilado de todos os testes realizados.

APÊNDICE D – RELATÓRIO DO TESTE DE USABILIDADE

Relatório do teste de usabilidade

Ferramenta para sincronização de conteúdos produzidos pelo Software QGIS

Madson Verdi Junior

Novembro de 2017

1 Empresa pampatec

1.1 Usuário 1

Após a instalação do plugin e execução inicial, um erro crítico foi apresentado. O ambiente foi restaurado e o teste teve de ser recomeçado

- **atualizar configurações pela interface gráfica**: Finalizada, erro médio;
- **atualizar configurações pelo arquivo de configuração**: Não finalizada
- **testar as configurações ativas**: Finalizada, sem erros
- **realizar a sincronização de dados**: Finalizada, com erro baixo;

2 Grupo de pesquisa

2.1 Usuário 1

- **atualizar configurações pela interface gráfica**: Finalizada, erro médio;
- **atualizar configurações pelo arquivo de configuração**: Finalizada, sem erros
- **testar as configurações ativas**: Finalizada, sem erros
- **realizar a sincronização de dados**: Finalizada, com erro médio;

2.2 Usuário 2

- **atualizar configurações pela interface gráfica**: Finalizada, erro médio;
- **atualizar configurações pelo arquivo de configuração**: Não finalizada
- **testar as configurações ativas**: Finalizada, sem erros
- **realizar a sincronização de dados**: Finalizada, com erro médio;

2.3 Usuário 3

- **atualizar configurações pela interface gráfica**: Finalizada, erro médio;
- **atualizar configurações pelo arquivo de configuração**: Não finalizada
- **testar as configurações ativas**: Finalizada, sem erros
- **realizar a sincronização de dados**: Finalizada, com erro baixo;

2.4 Usuário 4

- **atualizar configurações pela interface gráfica**: Finalizada, erro médio;
- **atualizar configurações pelo arquivo de configuração**: Não finalizada
- **testar as configurações ativas**: Finalizada, sem erros
- **realizar a sincronização de dados**: Finalizada, com erro baixo;

3 Conclusões

A tarefa `atualizar configurações pela interface gráfica` foi finalizada com erro médio pois a ferramenta apresentou problemas na troca de janelas.

A tarefa `atualizar configurações pelo arquivo de configuração` somente é realizada por usuários experientes e que realizaram o questionamento para o observador.

A tarefa `testar as configurações ativas` atingiu 100% de aproveitamento.

A tarefa `realizar a sincronização de dados` foi sempre finalizada com erros, porém os erros eram em sua maioria médio, pois a tarefa era sempre executada ao final, nos usuários classificados com erro de severidade baixa eles realizaram questionamentos sobre o erro anterior e tiveram de reiniciar o plugin para conseguirem finalizar a tarefa sem erro.

Anexos

ANEXO A – ARTIGO PUBLICADO: ERES 2017

Este artigo foi publicado na primeira edição da Escola Regional de Engenharia de Software no ano de 2017.

Uma Ferramenta para Sincronização de Conteúdos Produzidos pelo Software QGIS

Madson Verdi Junior, Claudio Schepke

[1]Universidade Federal do Pampa (UNIPAMPA) - Campus Alegrete
Av. Tiarajú, 810, 97546-550, Alegrete – RS – Brasil

madson.verdi@alunos.unipampa.edu.br, claudioschepke@unipampa.edu.br

Abstract. *Geographic Information System (GIS) is a computational resource that allows the management of information through data referenced from a coordinate system. Such systems store the information using two modes: files or Database (DB). The EIRE group stores energy resource data in QGIS (a GIS software) projects using files from a local server. To make the synchronization among local file data used by QGIS and a DB possible, it is necessary to implement a plugin for QGIS. This work describes the development of this plugin using software engineering techniques such as requirements analysis, prototyping, among others.*

Resumo. *Um* Sistema de Informações Geográficas *(SIG) é um recurso computacional que permite o gerenciamento de informações por meio de dados referenciados a partir de um sistema de coordenadas. Tais sistemas armazenam as informações utilizando dois modos: arquivos ou* Banco de Dados *(BD). O grupo EIRE armazena dados de recursos energéticos em projetos do SIG QGIS em arquivos de um servidor local. Para possibilitar a sincronização dos dados de arquivos locais utilizados pelo QGIS com um BD é necessário a implementação de um plugin para o SIG em questão. Este trabalho descreve o desenvolvimento deste plugin utilizando técnicas de* Engenharia de Software *como análise de requisitos, prototipagem, entre outras.*

1. Introdução

O *Grupo Exploração Integrada de Recursos Energéticos* (EIRE)[1] trabalha com a gestão de recursos energéticos utilizando um *Sistema de Informações Geográficas* denominado QGIS. A informação gerada e trabalhada pelo grupo está armazenada em arquivos. Estes arquivos estão armazenados em *Hard Drive*s externos ou diretamente nos computadores do grupo, e, em alguns casos, compartilhados em uma rede interna. A necessidade de um trabalho conjunto entre mais de um integrante do grupo de pesquisa exige a troca de informações geradas por cada membro. No modelo utilizado atualmente pelo grupo, a atualização contínua dos dados é dificultosa. O problema de compartilhamento

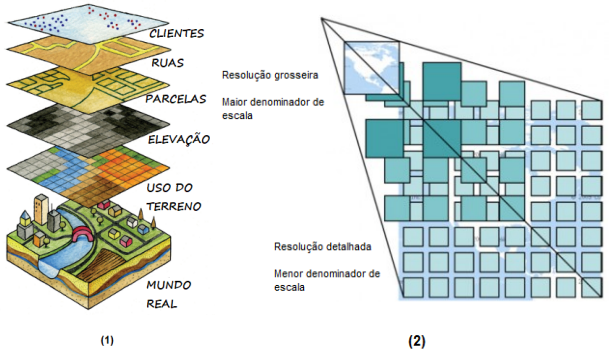
[1]*Grupo Exploração Integrada de Recursos Energéticos* ou EIRE é um grupo de pesquisa criado em 2011 e apresenta como principais objetivos: consolidar ações de ensino, pesquisa e extensão no campo do planejamento e gestão de recursos energéticos; desenvolver métodos e modelos para a simulação de gerenciamento de energia pelo lado da oferta e demanda; atuar na pesquisa e desenvolvimento de equipamentos de elevada eficiência energética; contribuir com ações de extensão e ensino para o desenvolvimento regional; pesquisa e desenvolvimento em fontes renováveis de energia.

de informação ocorre também em outros ambientes que utilizam o software para incluir conteúdo. Este problema, porém, só é observado quando mais de um usuário interage com o mesmo arquivo.

Este trabalho aborda a forma de centralizar a informação do software QGIS. O objetivo da criação do *plugin* é sincronizar os dados de forma fácil e clara, ressaltando também como a informação será armazenada em um banco de dados. Desta forma, os dados poderão ser facilmente difundidos em partes, ou totalmente, para outros grupos, facilitando o trabalho em conjunto, através de protocolos: WMS e WTMS.

A parte da esquerda da Figura 1 exemplifica a aplicação do padrão WMS onde as camadas, *CLIENTES*, *RUAS*, *PARCELAS*, *ELEVAÇÃO* e *USO DO TERRENO* representam os dados carregados de forma dinâmica, sobre um mapa real, representado no exemplo como a última camada, *MUNDO REAL*. Já a da parte direita da Figura 1 abstrai o padrão *Web Map Tile Service Implementation Standard* (WTMS), onde blocos podem ser carregados em diferentes níveis de detalhamento. Cada bloco de imagem pode conter as informações de todas camadas, já carregados em forma de imagens, essas divididas em pequenos pedaços.

Figura 1. Exemplo dos padrões (1) WMS (*Web Map Service Interface Standard*) e (2) WTMS (*Web Map Tile Service Implementation Standard*)



Este trabalho tem como objetivo prover um melhor aproveitamento dos dados catalogados pelo grupo de pesquisa EIRE. A motivação para o trabalho é a necessidade de centralizar as informações geradas de forma fácil e ágil. Isso é possível através da adoção de um banco de dados centralizado, onde a informação poderá ser acessada através de diversos dispositivos, incluindo móveis até computadores convencionais e, principalmente, garantir que todos os usuários do QGIS, envolvidos em um determinado projeto, possuam a informação atualizada. Para isso, as informações geradas através do software QGIS serão sincronizadas com um banco de dados que poderá ser acessado por todos envolvidos em determinado projeto.

2. Metodologia

2.1. Análise e Revisão de requisitos

Inicialmente foi necessário negociar e levantar os requisitos, conforme descrito por [Boehm et al. 1998]. A negociação é constituída pelas seguintes atividades:

1. identificação dos principais interessados no sistema ou subsistema;
2. determinação das "Condições de ganho" dos interessados;
3. negociação das condições gerais de ganho dos interessados para reconciliá-las em um conjunto de condições "ganha-ganha" para todos os envolvidos.

Com base nos requisitos apresentados acima e a descrição apresentada por [Pressman 2011], foi possível definir dois itens:

- **Análise de requisitos do problema**: Avaliar quais as tarefas que devem ser realizadas visando definir como proceder com o projeto;
- **Revisão dos requisitos analisados**: Após a análise dos requisitos é necessário avaliar se estes estão todos de acordo, conforme [Pressman 2011, p. 146-147]. Uma revisão nos requisitos pode identificar possíveis problemas a serem enfrentados e também gerar novos requisitos que não foram identificados anteriormente.

2.2. Implementação da solução

O *plugin*, também denominado *complemento*, implementado, é responsável pela sincronização dos dados entre cliente e servidor. O servidor irá processar as requisições geradas por este *plugin*, a fim de manter os dados atualizados entre o cliente e o servidor.

Para a implementação, os seguintes passos foram executados:

1. **Protótipos de tela para o *plugin***: A prototipação de uma solução, segundo [Pressman 2011, p.62], é necessária quando o desenvolvedor não se sente seguro na interação homem/máquina e esta interação necessita ser detalhada.
2. **Geração das consultas**: o comando *shp2pgsql* [Ramsey et al. 2005] possibilita gerar os arquivos com a extensão *.sql* a partir dos dados dos arquivos com a extensão *.shp*. O oposto é realizado através do comando *pgsql2shp* [Ramsey et al. 2005].
3. **Servidor da aplicação**: o servidor recebe as requisições e encaminha respostas para um *cliente*. Seu acesso será realizado através de PostGIS, que é uma extensão para PostgreSQL. Nele é possível armazenar as informações no banco de dados. O acesso ao banco de dados é realizado quando o computador estiver conectado a uma rede onde o servidor esteja visível.
4. ***Plugin***: funciona como uma interface para o usuário utilizar a solução. Este está diretamente ligado ao *Software QGIS*, funcionando como um complemento, adicionando a funcionalidade proposta neste trabalho.

3. Resultados

Os resultados gerados foram divididos em alguns grupos, descrito na apresentação da metodologia, e são discriminados nas sub-seções abaixo.

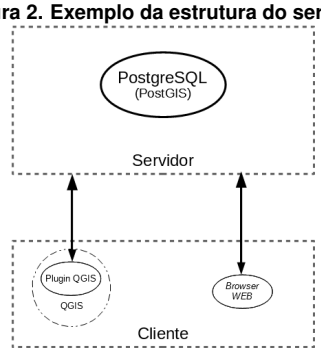
3.1. Análise e Revisão de requisitos

Durante o processo de análise de requisitos foi possível identificar a necessidade de implementação das seguintes funcionalidades:

- cliente (*plugin* para o QGIS):
  - capturar os eventos ocorridos na edição de dados de um projeto no QGIS;
  - enviar as alterações para o servidor;
  - receber as atualizações do servidor;
  - o cliente não possui banco de dados.
- cliente (*borwsers*):
  - visualizar mapas através de *browsers* na WEB e intranet;
  - a visualização deve possuir controle de acesso.
- servidor:
  - armazenar os mapas atualizados;
  - receber atualização de clientes;
  - enviar atualização para clientes.
  - WEB - QGIS Server:
    - permite a utilização dos padrões WMS e WTMS.

Uma simplificação desta análise pode ser visualizada na Figura 2.

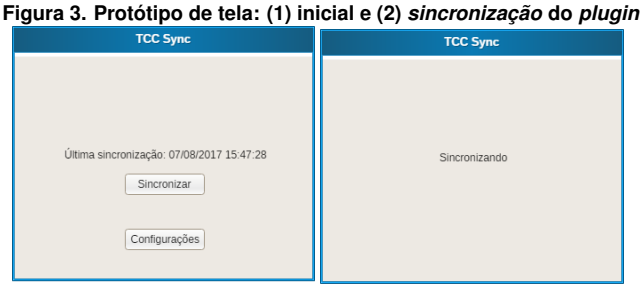
Figura 2. Exemplo da estrutura do servidor



Durante a revisão dos requisitos levantados, conforme descrito anteriormente nesta seção, foi possível identificar alguns desafios:

- a sincronização deverá ocorrer quando o *plugin* for carregado;
  - para realizar o envio, o arquivo deverá ser convertido para o formato *.sql*;
  - para realizar o recebimento, o arquivo deverá ser convertido para o formato *.shp*.
- o servidor deverá armazenar todos os scripts recebidos e repassar para todos clientes, se for necessário;
- o cliente deverá manter uma data de atualização para verificar qual ou quais scripts devem ser retornados do servidor;
- o nome do banco deverá ser armazenado junto ao projeto;
- o servidor deverá receber o script e atualizar/criar as suas bases de dados;
- os usuários necessitam de níveis de acesso.

Figura 3. Protótipo de tela: (1) inicial e (2) *sincronização* do *plugin*

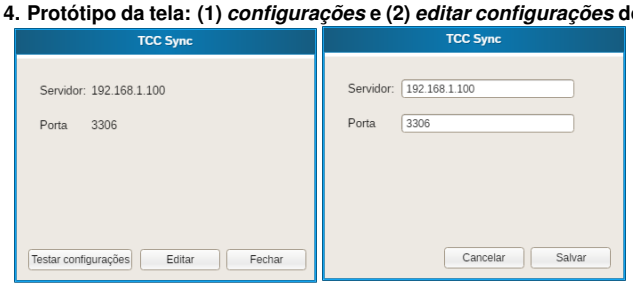


3.2. Prototipação da solução

Os protótipos de tela gerados para exemplificar a interação homem/máquina foram gerados utilizando o software *Pencil Project*[2].

A parte (1) da Figura 3 representa o protótipo da tela inicial do *plugin*, ao clicar em *Sincronizar* espera-se que seja carregada a tela descrita na parte (2) da Figura 3 e ao final da sincronização deve retornar para a parte (1). Caso tenha ocorrido erro no processo uma mensagem de erro deve ser exibida no local da data da última atualização.

Figura 4. Protótipo da tela: (1) *configurações* e (2) *editar configurações* do *plugin*



A opção *Configurações*, disponível na tela inicial, deve carregar a tela descrita na parte (1) da Figura 4. Nesta tela deverão ser listadas todas as configurações que o *plugin* necessita para realizar a conexão com o servidor, permitindo que o usuário altere essas configurações de forma visual, conforme a parte (2) da Figura 4. As informações de configuração devem ser carregadas e armazenadas no arquivo de configuração do *plugin*.

3.3. Implementação da solução

A implementação da solução ocorreu em duas fases, primeiro a fase de validação da ideia e em um segundo momento a implementação da solução por completo.

[2]É um sofware de código aberto sobre a licença *GNU Public License version 2* [License 1989] e pode ser baixado gratuitamente pelo link https://pencil.evolus.vn/Downloads.html

3.3.1. Primeira fase

Durante a implementação da solução do cliente foi possível identificar que um arquivo em formato *.shp* com aproximadamente 10 MB quando convertido para formato *.sql* ultrapassava os 35 MB. Desta forma buscou-se gerar arquivos menores o que por sua vez gerou os seguintes desafios:

- **como gerar arquivos menores**: utilizando um padrão utilizado em sistemas de versionamento, encaminhando somente as alterações realizadas nos arquivo. Para isso duas alternativas foram encontradas:
  - em sistemas *Unix* o comando nativo *diff* realizava a tarefa sem problemas, porém em sistemas *Windows* não era uma alternativa e logo foi descartado;
  - o *Python*[3] possui uma biblioteca, *difflib* [van Rossum and Drake 2017], com o mesmo propósito de gerar a diferença entre um arquivo de entrada e outro de saída.
- **como enviar**: é necessário encaminhar o resultado da diferença para o servidor, o Script 2 realiza a conversão para o formato *.sql*;
- **como receber**: o *Cliente* recebe somente as atualizações que são mescladas com seu arquivo original e convertidas novamente para o formato *.shp*, a conversão é realizada através do Script 3;
- **processamento do arquivo recebido**: após o *Cliente* receber os comandos *.sql* de atualização eles são adicionados em um arquivo já gerado. Posteriormente este arquivo é convertido para *.shp*, nesta etapa o arquivo de configuração é gerado e um exemplo pode ser visualizado no Script 1.

A Figura 2 mostra a estrutura montada entre cliente e servidor, exemplificando a troca de informações entre as partes, onde o cliente encaminha os dados e o servidor recebe. As informações são atualizadas no servidor, assim o próximo cliente que tentar realizar a sincronização deverá, primeiramente, atualizar os dados locais conforme as informações armazenadas no servidor. O script em *Python* gera um novo arquivo local com as modificações realizadas no cliente para ser encaminhado para o servidor.

As consultas são geradas pelo cliente da aplicação e convertidas para o formato local correto, um arquivo de configuração é criado no cliente para auxiliar no gerenciamento dos arquivos que estão sendo trabalhados, um trecho deste arquivo de configuração pode ser verificado no Script 1.

```
1 [Rodovias_RioGrandeDoSul_2015_1]
2 local = 2017-06-19 14:50:49.938528
3 file_original = ../arquivos_entrada/MapaRodoviarioRS/Rodovias_RioGrandeDoSul_2015_1.shp
4 server = 2017-06-19 14:50:50.758714
5 sql_original = ../arquivos_saida/Rodovias_RioGrandeDoSul_2015_1_original.sql
6 file_temp = ../arquivos_saida/Rodovias_RioGrandeDoSul_2015_1.shp
7 sql_updated = ../arquivos_saida/Rodovias_RioGrandeDoSul_2015_1.sql
```

Script 1. Trecho de arquivo de configuração gerado após atualização

O servidor da aplicação consiste em um servidor com um banco de dados PostgreSQL com a extensão PostGIS.

O cliente foi implementado em *Python* para facilitar a implementação de um *plugin* para o QGIS. A implementação do cliente roda basicamente em cima de duas funções.

[3]Python é uma linguagem de programação de alto nível [Venners 2003], interpretada, de script, imperativa, orientada a objetos, funcional, de tipagem dinâmica e forte.

A primeira, descrita no Script 2, realiza a conversão do arquivo binário em *.shp* para o formato *.sql*. A segunda função, descrita no Script 3, recupera a informação armazenada no servidor e gera o arquivo *.shp*.

```
def create_sql(bin_folder, input_file, output_files_folder, filename, sufix = ''):
  # nomes de arquivos de origem e atual - SQL
  input_sql_file_name = output_files_folder + DS + filename + sufix + '.sql'
  # print input_sql_file_name
  # gerando arquivos para comparacao
  args_input = bin_folder + DS + 'shp2pgsql.exe -W "latin1" -e ' + input_file + ' > ' + input_sql_file_name

  # print args_input

  try:
    retcode = subprocess.call(args_input, shell=True)
    if retcode < 0:
      print >>sys.stderr, "Child was terminated by signal", -retcode
    else:
      print >>sys.stderr, "Child returned", retcode
  except OSError as e:
    print >>sys.stderr, "Execution failed:", e

  return input_sql_file_name
```

Script 2. Conversão de .shp para .sql

```
def create_shp(bin_folder, output_files_folder, filename, sufix = ''):
  # nomes de arquivos de origem e atual - SQL
  shp_file_name = output_files_folder + DS + filename + sufix
  # print input_sql_file_name
  # gerando arquivos para comparacao
  args_input = bin_folder + DS + 'pgsql2shp.exe -f "' + shp_file_name + '" -h ' + config_get('SERVER', 'pg_host') + \
    ' -P ' + config_get('SERVER', 'pg_pass') + \
    ' -u ' + config_get('SERVER', 'pg_user') + \
    ' ' + config_get('SERVER', 'pg_database') +\
    ' ' + filename.lower()

  print args_input

  try:
    retcode = subprocess.call(args_input, shell=True)
    if retcode < 0:
      print >>sys.stderr, "Child was terminated by signal", -retcode
    else:
      print >>sys.stderr, "Child returned", retcode
  except OSError as e:
    print >>sys.stderr, "Execution failed:", e

  config_set(filename, 'file_temp', shp_file_name + '.shp')

  return shp_file_name
```

Script 3. Conversão de .shp para .sql

A fim de realizar a validação da implementação da solução, uma bateria de testes foi realizada utilizando duas VMs[4] , assim, permitindo uma simulação entre 2 clientes. Sendo que o **Cliente 1** é o computador Host das VMs. O seguinte cenário foi executado:

1. Cliente 1 gera um novo mapa e sincroniza com servidor;
2. Cliente 2 realiza o recebimento desse novo mapa;
3. Cliente 2 atualiza localmente o mapa;
4. Cliente 2 realiza sincronização com o servidor;
5. Cliente 1 atualiza localmente o mapa;
6. Cliente 1 realiza sincronização com o servidor;
   (a) Recebe *.sql* do servidor;
   (b) Gera *.sql* localmente com base no arquivo do mapa;
   (c) Gera o arquivo de diferença do arquivo *.sql*;
   (d) Executa o script *.sql* no servidor, somente com as modificações geradas;

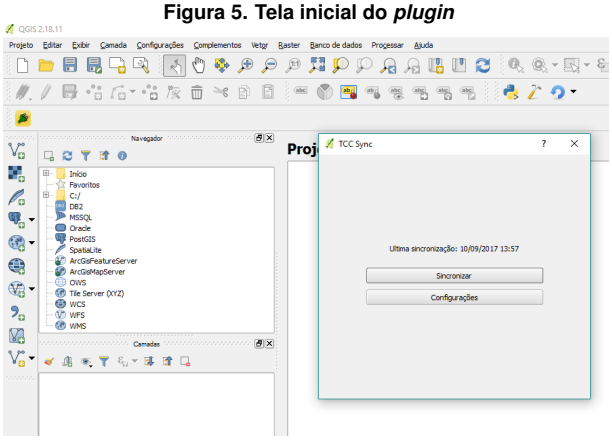
A solução implementada se comportou conforme o esperado, o que possibilitou a implementação da segunda fase do projeto, descrita na Seção 3.3.2.

[4]Uma máquina virtual (*Virtual Machine ou Máquina Virtual* – VM) pode ser definida como "uma duplicata eficiente e isolada de uma máquina real".

3.3.2. Segunda fase

A segunda fase da implementação da solução consistiu em gerar uma interface para o usuário interagir com a aplicação. A interface foi implementada seguindo o modelo prototipado e suas interações.

Figura 5. Tela inicial do *plugin*



A Figura 5 é referente a tela inicial do *plugin* implementado, onde os botões *Sincronizar* e *Configurações* redirecionam para suas devidas funcionalidades.

4. Considerações finais e Trabalhos futuros

A implementação do *plugin* foi finalizada e encontra-se na etapa de testes com o usuário final. Agora é necessário um agendamento com o EIRE para dar continuidade ao trabalho O software também será validado utilizando alguma empresa do *Parque Científico e Tecnológico do Pampa* (PampaTec).

Referências

Boehm, B., Egyed, A., Kwan, J., Port, D., Shah, A., and Madachy, R. (1998). Using the winwin spiral model: a case study. *Computer*, 31(7):33–44.

License, G. G. P. (1989). Version 2, june 1991. *Copyright (C)*, 1991:02111–1307.

Pressman, R. S. (2011). Engenharia de software: uma abordagem profissional. 7ª edição. *Ed: McGraw Hill*.

Ramsey, P. et al. (2005). Postgis manual. *Refractions Research Inc*.

van Rossum, G. and Drake, F. L. (2017). disponível livermente na internet em http://www. python. org. br/wiki.

Venners, B. (2003). The making of python. *Artima. com. http://www. artima. com/intv/python. html [accessed 2003-12-09]*.