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 Driver*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

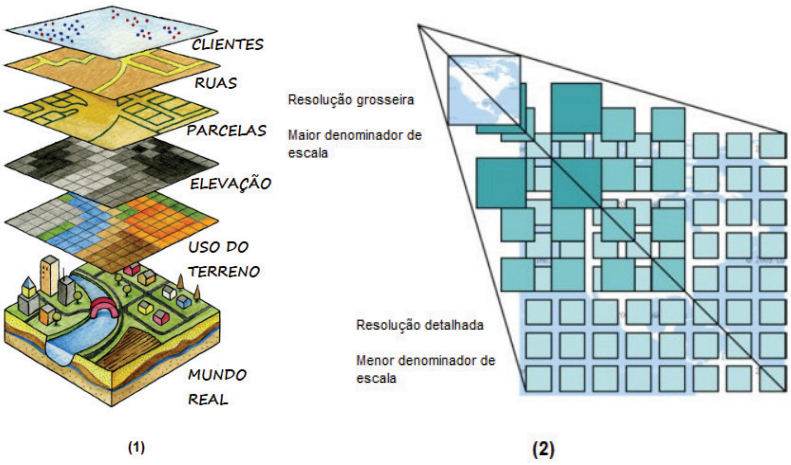
[1]*Grupo Exploração Integrada de Recursos Energéticos* ou EIRE é um grupo de pesquisa criado em 2011 e apresenta como principais objetivos: consolidar ações de ensino, pesquisa e extensão no campo do planejamento e gestão de recursos energéticos; desenvolver métodos e modelos para a simulação de gerenciamento 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.

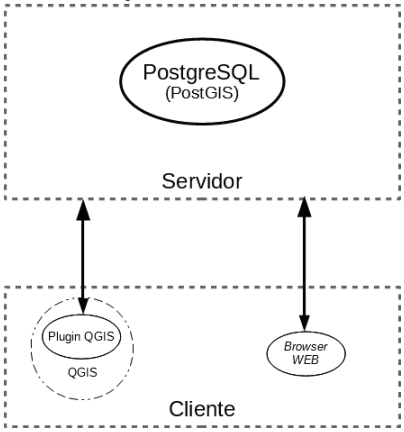
3.1. Análise e Revisão de requisitos

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.

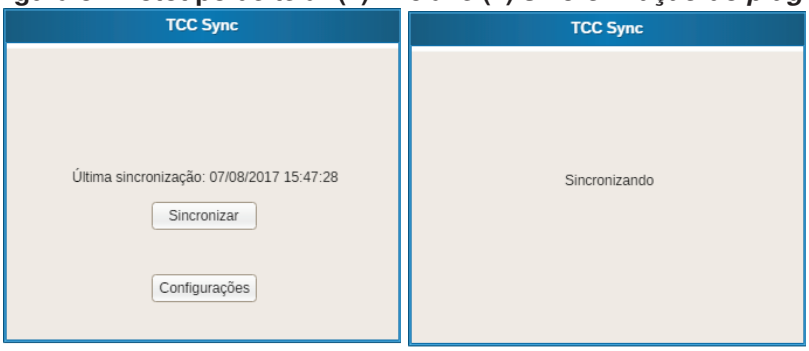
Figura 2. Exemplo da estrutura do servidor



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*

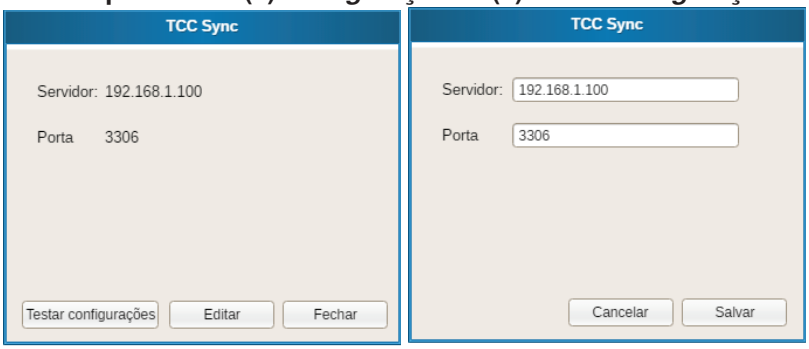


3.2. Prototipação da solução

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;

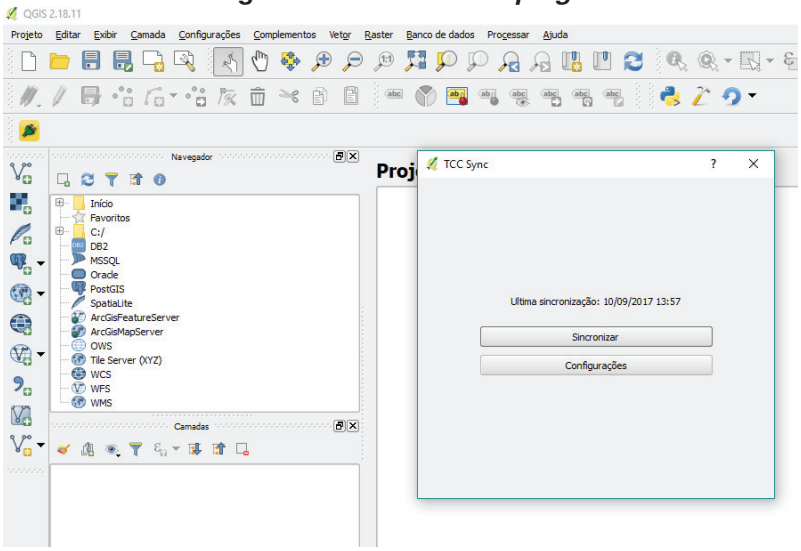
A solução implementada se comportou conforme o esperado, o que possibilitou a implementação da segunda fase do projeto, descrita na Seção 3.3.2.

[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 livremente 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]*.