UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
PPGE3M – PROGRAMA DE PÓS GRADUAÇÃO EM ENGENHARIA DE MINAS, METALÚRGICA E DE MATERAIS

MMD00580 – Programação Científica em Python

Ronald Scheffer Leal

INTEGRAÇÃO DAS LINGUAGENS DE PROGRAMAÇÃO JAVASCRIPT E PYTHON PARA O SOFTWARE STUDIO 3

PORTO ALEGRE
2016

INTRODUÇÃO

Este *script* possui a finalidade de integrar as linguagens de programação *JavaScript* e *Python* para utilização no *software Studio 3*. O motivo de tal desenvolvimento foi de conseguir aproveitar os diversos recursos e bibliotecas existentes em *Python* e aplica-los nos códigos desenvolvidos.

É sabido que o *Studio 3* possui diversas limitações na parte de script, que vão desde uma *engine* bastante defasada (utiliza-se o Internet Explorer versão 7 como programa para execução dos códigos) até dificuldades em se conhecer todos os métodos e propriedades dentro da linguagem de programação *Datamine API*, que é a biblioteca de funções do *software*. Por este motivo, a utilização das bibliotecas disponíveis em *Python* poderá dar um ganho enorme para os *scripts* desenvolvidos dentro do *Studio 3*.

Para esta implementação, foram testadas uma série de alternativas que na sua grande maioria não funcionaram devido a *engine* do *software*. Finalmente, encontrou-se a solução de utilizar um aplicativo chamado *Python COM Server*, que funciona como um servidor que registra dentro do sistema operacional as classes e funções escritas em *Python*. Este registro, dentro do *Windows*, é conhecido como um objeto do tipo *ActiveXObject*, que é o mesmo tipo de objeto que se invoca para acessar a biblioteca de funções do *Studio 3*. O fato de já se conhecer a forma de importar a biblioteca de funções do *software* foi fundamental para se chamar as funções registradas para dentro do código *JavaScript*.

A partir do momento que o *Python COM Server* foi importado para o código *JavaScript*, é possível acessar todas as suas funções e propriedades. A conexão entre as duas linguagens de programação se dá pelo servidor na forma de envio e retorno de objetos no formato de *strings* (podem ser utilizados outros formatos para envio de objetos, mas as *strings* são mais aconselháveis por possuir o mesmo formato de leitura nas duas linguagens), que sempre são lidas e traduzidas de *string* de volta para objeto no formato da linguagem que está recebendo a variável.

Para a correta funcionalidade do *Python COM Server*, é necessário que o usuário possua instalado as bibliotecas *Python* e *PyWin32*, que é a biblioteca que permite o registro destas funções como um objeto lido pelo *Windows*. A Figura 1 mostra um exemplo básico do servidor de funções *Python*.

```python
class ShowMeDoDemo:
    _public_methods_ = [ 'Hello' ]
    _reg_progid_ = "ShowMeDo.Demo"
    _reg_desc_ = "Python Test COM Server"
    import pythoncom
    _reg_clsid_ = str(pythoncom.CreateGuid()) # e.g. "{4D93DCBC-DE99-4D5A-ACFC-44F82BDB9889}"

    def Hello(self):
        return "Hello World!"

if __name__ == "__main__":
    # use 'python com.py' to register the COM server
    # use 'python com.py --unregister' to unregister it
    print "Registering COM server..."
    import win32com.server.register
    win32com.server.register.UseCommandLine(ShowMeDoDemo)
```

Figura 1 - Exemplo do Python COM Server, onde se pode observar a criação e registro do servidor, das classes e funções em Python.

FUNCIONALIDADE

Foi desenvolvido um s*cript* teste com algumas operações que são bastante utilizadas no desenvolvimento de *scripts* no *Studio 3*. A finalidade deste código não era de implementar uma função nova, mas sim testar a comunicação entre as duas linguagens através do servidor. Para isso, algumas funções que já haviam sido escritas em *JavaScript* foram traduzidas para *Python*.

O *script* desenvolvido calcula os blocos vizinhos a partir de um modelo de blocos regular importado como *input* no código. Esta é uma função bastante útil em códigos que envolvam sequenciamento de lavra e cálculo de diluição, e para a realização destes testes foram escritas duas funções em Python. A Figura 2

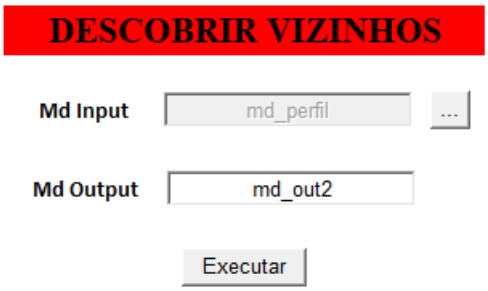

Figura 2 - Menu principal do script.

A primeira função foi chamada de infoproto e tinha a finalidade de calcular a última linha, coluna e bancada do modelo de blocos, de forma a reconhecer onde o modelo de blocos terminava e não calcular algum vizinho para além dos seus limites. A função recebeu como argumento três números do *JavaScript* que foram reconhecidos como *string* dentro do *Python*. Após os cálculos, a função retornou um *array* na forma de *string* para o *JavaScript*.

A segunda função calculava os vizinhos a partir do número identificador (*IJK*) de cada bloco. Ela recebeu como argumento as informações calculadas na função anterior e um vetor (*string*) com todos os blocos do modelo, retornando uma *string* com todos os vizinhos calculados.

Ambas as funções puderam ser chamadas via *JavaScript* executado dentro do painel de *scripts* do *software Studio 3* e funcionaram corretamente, retornando os valores esperados. Não foi observada nenhuma limitação na questão de recebimento e envio de dados entre o *JavaScript* e o *Python*.

Espera-se que trabalhos futuros possam ser desenvolvidos a partir da mistura das duas linguagens de programação. A utilização de bibliotecas de funções *Python* como o *numpy* e o *scipy* poderá abrir um leque de novas oportunidades de desenvolvimento de códigos dentro do *Studio 3*.

Abaixo seguem os códigos desenvolvidos e comentados nas linguagens *JavaScript* e *Pyhton*.

```
CODIGO JAVASCRIPT

<html>
    <head>

        <style type="text/css">
            .tabela{
                width: 300;
                font-family: Calibri;
                font-size: 14px;
                text-align: center;
                font-weight: bold;
            }

            .textocentralizado{
                text-align: center;
            }

            .assinatura{
                text-align: right;
                font-size: 11px;
                font-family: Calibri;
                font-weight: bold;
            }

            .titulo{
                width: 300;
                text-align: center;
                font-weight: bold;
                font-size: 22px;
                background-color: red;
            }
        </style>

        <script type="text/javascript" src="json2.js"></script>
        <script type="text/javascript">

            //VARIAVEIS DE INICIALIZACAO DAS BIBLIOTECAS DATAMINE E PYTHON
            var oScript = null;
            var oDmApp = null;
            var oPycom = null;

            //FUNCAO NECESSARIA PARA IMPORTAÃ‡Ãƒo DA BIBLIOTECA DO DATAMINE E DO SERVIDO PYTHO
            function AutoConnect(){
                try{
                    oScript = new ActiveXObject("StudioCommon.ScriptHelper");
                }
                catch(e){
                    oScript = new ActiveXObject("DatamineStudio.ScriptHelper");
                }
                oScript.initialize(window);
                oDmApp = oScript.getApplication();

                oPycom = new ActiveXObject("CalcVizinhos");
            }

            //FUNCAO PRINCIPAL
            function execute(){
                //CRIA AS COLUNAS DE VIZINHOS NO ARQUIVO DO MODELO DE BLOCOS
                oDmApp.ParseCommand("extra &IN=" + mdblocosin.value + " &OUT=" + mdblocosout.v
                 " 'VNORTE=0'"+
                 " 'VSUL=0'"+
                 " 'VLESTE=0'"+
                 " 'VOESTE=0'"+
                 " 'VCIMA=0'"+
                 " 'VBAIXO=0'"+
                 " 'GO'");

                //ABRE O ARQUIVO E LOCALIZA O NUMERO DAS COLUNAS DE VIZINHO CRIADAS
```

```
70.                 var dmHandler = new ActiveXObject("DmFile.DmTableADO");
71.                 dmHandler.Open(oDmApp.ActiveProject.GetDBObjFilePath(mdblocosout.value),false)
72.                 nrcolunas = dmHandler.Schema.FieldCount;
73.                 for(var i = 0; i < nrcolunas; i++){
74.                     if(dmHandler.Schema.GetFieldName(i+1) == "VNORTE"){
75.                         var nrcolnorte = i + 1;
76.                         var nrcolsul = i + 2;
77.                         var nrcolleste = i + 3;
78.                         var nrcoloeste = i + 4;
79.                         var nrcolcima = i + 5;
80.                         var nrcolbaixo = i + 6;
81.                     }
82.                 }
83.
84.                 //COLETA AS INFORMACOES NECESSARIAS
85.                 dmHandler.MoveLast();
86.                 var ny = dmHandler.getNamedColumn("NY");
87.                 var nz = dmHandler.getNamedColumn("NZ");
88.                 var ijkultimobloco = dmHandler.getNamedColumn("IJK");
89.
90.                 //CHAMA A FUNCAO PYTHON QUE CALCULA O I, J, K MAXIMO DO MODELO DE BLOCOS
91.                 var ijkmax = oPycom.infoproto(ny,nz,ijkultimobloco);
92.
93.                 //GUARDA TODOS OS VALORES DE IJK DO ARQUIVO DO MODELO DE BLOCOS EM UM VETOR E
94.                 var vetorijk = [];
95.                 dmHandler.MoveFirst();
96.                 while(!dmHandler.EOF){
97.                     vetorijk.push(dmHandler.getNamedColumn("IJK"));
98.                     dmHandler.MoveNext();
99.                 }
100.                vetorijk = "[" + vetorijk + "]";
101.
102.                //CHAMA A FUNCAO PYTHON QUE CALCULA OS VIZINHOS E LOGO EM SEGUIDA JA CHAMA A F
103.                var vetorvizinhos = oPycom.calcvizinhos(ny,nz,vetorijk,ijkmax);
104.                vetorvizinhos = transformardados(vetorvizinhos);
105.
106.                //ESCREVE O NUMERO DOS VIZINHOS NO ARQUIVOS
107.                dmHandler.MoveFirst();
108.                for(var i = 0; i < vetorvizinhos.length; i = i + 6){
109.                    dmHandler.SetColumn(nrcolnorte,vetorvizinhos[i]);
110.                    dmHandler.SetColumn(nrcolsul,vetorvizinhos[i+1]);
111.                    dmHandler.SetColumn(nrcolleste,vetorvizinhos[i+2]);
112.                    dmHandler.SetColumn(nrcoloeste,vetorvizinhos[i+3]);
113.                    dmHandler.SetColumn(nrcolcima,vetorvizinhos[i+4]);
114.                    dmHandler.SetColumn(nrcolbaixo,vetorvizinhos[i+5]);
115.                    dmHandler.MoveNext();
116.                }
117.                dmHandler.Close();
118.            }
119.
120.            //ESTA FUNCAO RECEBE O VETOR DO PYTHON COMO UMA STRING E TRANSFORMA O MESMO EM UM
121.            function transformardados(vetor){
122.                vetor2 = [];
123.                try{
124.                    vetor2 = JSON.parse(vetor);
125.                }
126.                catch(e){
127.                    var vetor = vetor.replace("[","").replace("]","").split(",");
128.                    for(var i = 0; i < vetor.length; i++){
129.                        vetor2[i] = Number(vetor[i]);
130.                    }
131.                }
132.                return vetor2;
133.            }
134.
135.            //ABRE O BROWSER DENTRO DO DATAMINE PARA ESCOLHA DO MODELO DE BLOCOS DE ENTRADA
136.            function buscararquivo(nome,tipofiltro){
137.                var browser=oDmApp.ActiveProject.Browser;
138.                browser.TypeFilter=oScript.DmFileType[tipofiltro];
139.                browser.show(false);
```

```
140.                 nome.value=browser.FileName;
141.             }
142.
143.         </script>
144.     </head>
145.     <body onload="AutoConnect();" onunload="oDmApp = null;oScript = null;">
146.
147.         <table border="0" class="titulo">
148.             <tr>
149.                 <td> DESCOBRIR VIZINHOS </td>
150.             </tr>
151.         </table> <br>
152.
153.         <table border="0" class="tabela">
154.             <tr>
155.                 <td width="100"> Md Input </td>
156.                 <td> <input id="mdblocosin" class="textocentralizado" disabled> </td>
157.                 <td width="40"> <input type="button" value="..." style="width:25" onclick="bus
158.             </tr>
159.         </table> <br>
160.
161.         <table border="0" class="tabela">
162.             <tr>
163.                 <td width="100"> Md Output </td>
164.                 <td> <input id="mdblocosout" class="textocentralizado"> </td>
165.                 <td width="40"> </td>
166.             </tr>
167.         </table> <br>
168.
169.         <table border="0" class="tabela">
170.             <tr>
171.                 <td> <input type="button" value="Executar" onclick="execute()"> </td>
172.             </tr>
173.         </table>
174.
175.     </body>
176. </html>
```

```
CODIGO PYTHON

#DEFINICAO DA CLASSE E FUNCOES DO SERVIDOR E IMPORTACAO DAS BIBLIOTECAS
class vizinhos:
  _public_methods_ = [ 'infoproto', 'calcvizinhos' ]
  _reg_progid_ = "CalcVizinhos"
  _reg_desc_ = "Python Test COM Server"
  import pythoncom
  _reg_clsid_ = str(pythoncom.CreateGuid()) # e.g. "{4D93DCBC-DE99-4D5A-ACFC-44F82BDB9889}"

  #ESTA FUNCAO RECEBE COMO ARGUMENTO O NUMERO DE BLOCOS EM Y E EM Z E O VALOR DO IJK DO ULTIMO
  #DO JAVASCRIPT E RETORNA O MAXIMO I, J E K POSSIVEIS
  def infoproto(self,ny,nz,ijkultimobloco):
    import math

    #TRANSFORMA EM NUMERO OS ARGUMENTOS STRING QUE FORAM RECEBIDOS DO JAVASCRIPT
    ny = int(ny)
    nz = int(nz)
    ijkultimobloco = int(ijkultimobloco)

    #CALCULO DO MAXIMO I, J E K POSSIVEL
    imax = math.floor(ijkultimobloco/(ny * nz)) + 1
    jmax = math.floor((ijkultimobloco - (imax - 1) * ny * nz)/ nz) + 1
    kmax = ijkultimobloco - ((imax - 1) * ny * nz) - ((jmax - 1) * nz) + 1

    #ARMAZENA OS CALCULOS EM UM VETOR E RETORNA O MESMO PARA O JAVASCRIPT
    ijkmax=[imax,jmax,kmax]
    return str(ijkmax)

  #ESTA FUNCAO RECEBE COMO ARGUMENTO O NUMERO DE BLOCOS EM Y E EM Z E UM VETOR CONTENDO TODOS
  #DE BLOCOS E RETORNA OS VIZINHOS DE CADA UM DOS MODELOS
  def calcvizinhos(self,ny,nz,vetorijk2,ijkmax2):
    import math

    #TRANSFORMA EM NUMERO E VETOR OS ARGUMENTOS RECEBIDOS DO JAVASCRIPT
    vetorijk = eval(vetorijk2)
    ijkmax = eval(ijkmax2)
    imax = int(ijkmax[0])
    jmax = int(ijkmax[1])
    kmax = int(ijkmax[2])
    ny = int(ny)
    nz = int(nz)

    ijkvizinhos = []
    for ijkbloco in vetorijk:
      #CALCULA O I, J E K DO BLOCO EM ANALISE, PARA CASO ELE SEJA DE BORDA DESCONSIDERE POSSIV
      ibloco = math.floor(ijkbloco/(ny * nz)) + 1
      jbloco = math.floor((ijkbloco - (ibloco - 1) * ny * nz)/ nz) + 1
      kbloco = ijkbloco - ((ibloco - 1) * ny * nz) - ((jbloco - 1) * nz) + 1

      #RESETA OS VALORES DOS IJK DOS VIZINHOS PARA A PROXIMA ITERACAO
      vizinhonorte = -99
      vizinhosul = -99
      vizinholeste = -99
      vizinhooeste = -99
      vizinhoemcima = -99
      vizinhoembaixo = -99

      #CALCULA O IJK DOS VIZINHOS, RESPEITANDO AS SITUACOES DE BLOCOS DE BORDA
      if(jbloco != jmax):
        vizinhonorte = ijkbloco + nz
      if(jbloco != 1):
        vizinhosul = ijkbloco - nz
      if(ibloco != imax):
        vizinholeste = ijkbloco + (ny * nz)
      if(ibloco != 1):
        vizinhooeste = ijkbloco - (ny * nz)
      if(kbloco != kmax):
        vizinhoemcima = ijkbloco + 1
```

```
70.         if(kbloco != 1):
71.           vizinhoembaixo = ijkbloco - 1
72.
73.         #ARMAZENA NO FINAL DO VETOR O IJK DOS VIZINHOS
74.         ijkvizinhos.append(vizinhonorte)
75.         ijkvizinhos.append(vizinhosul)
76.         ijkvizinhos.append(vizinholeste)
77.         ijkvizinhos.append(vizinhooeste)
78.         ijkvizinhos.append(vizinhoemcima)
79.         ijkvizinhos.append(vizinhoembaixo)
80.
81.       #RETORNA O VETOR DE VIZINHOS PARA O JAVASCRIPT
82.       return str(ijkvizinhos)
83.
84.  #REGISTRO DO SERVIDOR NO SISTEMA
85.  if __name__ == "__main__":
86.    # use 'python com.py' to register the COM server
87.    # use 'python com.py --unregister' to unregister it
88.    print "Registering COM server..."
89.    import win32com.server.register
90.    win32com.server.register.UseCommandLine(vizinhos)
```