IEGULDĪJUMS TAVĀ NĀKOTNĒ!

Eiropas Reģionālās attīstības fonds

Prioritāte: 2.1. Zinātne un inovācijas

Pasākums: 2.1.1. Zinātne, pētniecība un attīstība

Aktivitāte: 2.1.1.1. Atbalsts zinātnei un pētniecībai

Projekts: ”Multi - modeļu izstrādes tehnoloģija .NET pielietojumu projektiem”

Projekta sākuma datums: 2014.gada 1.janvāris.

Projekta beigu datums: 2015.gada 30.jūnijs.

Līguma Nr. 2013/0031/2DP/2.1.1.1.0/13/APIA/VIAA/010

ESF finansējuma saņēmējs: SIA, SWH SETS

Sadarbības partneris: Elektronikas un datorzinātņu institūts (EDI)

Projekta aktivitātes Nr. 5.5 "Datu apstrādes funkciju bibliotēkas integrēšana" progresa pārskats

Pārskats Nr. 38 par periodu no 2015.gada 1.janvāra līdz 2015.gada 30.jūnijam.

SATURS

1. Kopsavilkums

Pārskata periodā (2015-01-01 – 2015-06-30) projekta „Multi - modeļu izstrādes tehnoloģija .NET pielietojumu projektiem” aktivitātes "Datu apstrādes funkciju bibliotēkas integrēšana" ietvaros veikti šādi darbi:

1. Ārējo bibliotēku datu apstrādes funkciju integrācijas shēmas analīze.
2. Ārējo bibliotēku datu apstrādes funkciju integrācijas shēmas izstrāde, kas ietver RSimportMethod, RSProcMethod un RSexportMethod funkciju saimju reģistrācijas un izpildes nodrošinājuma izstrādi.
3. Aktivitātes pētnieciskā darbība apspriesta ik nedēļas projekta semināros.

2. Ievads

Šī zinātniskā atskaite ir veltīta projekta apakšaktivitātes Nr.3.9.1. " Datu apstrādes funkciju bibliotēkas integrēšana" ietvaros veiktajai izstrādei pārskata periodā (2015-01-01. – 2015-06-30.).

Paraugprojekta specifikācijā [1] ir paredzētas vairākas biznesa funkcijas, kas apstrādās projekta datu objektus. Bet vairums no tām atrodas jau izstrādātajās bibliotēkās, kuras ir plānots dinamiski ielādēt sistēmā un ļaut lietotājam izsaukt no tām funkcijas. Jo vairāk - ir vēlme nākotnē papildināt šo funkciju klāstu.
Šim nolūkam tika izstrādāta infrastruktūra, kas ļaus lietotājam, neaiztiekot sistēmu, piereģistrēt ārējās datu apstrādes funkcijas, izsaukt tās un saglabāt rezultātus datu bāzē.

3. Datu apstrādes funkciju reģistrs

Lai varētu operēt ar nezināmām funkcijām, kuru klāsts tiks papildināts dinamiski, ir paredzēti speciālie objekti, kuros apraksta datu apstrādes funkcijas: identifikāciju (nosaukumu un bibliotēku), parametrus un atgriežamos rezultātus. Šādi objekti ir trīs veidu - atkarībā no nolūka. Katram veidam ir paredzēts, ar kādiem parametru un rezultātu tipiem principā varētu strādāt, un kāds ir darbības mērķis (respektīvi, ja grib nodefinēt jaunu funkciju, ir jāizvēlas, kuram veidam tā pieder):

- **RSimportMethod** - funkcija domāta viena objekta importam no norādītā faila (var izvēlēties, kādu objekta tipu funkcija atgriezīs). Importējamais objekts tiks piereģistrēts datu bāzē.
- **RSprocMethod** - funkcija domāta piereģistrētā objekta (vai vairāku objektu) apstrādei, kuras rezultātā rodas viens vai vairāki datu objekti. Rezultātus piereģistrē datu bāzē.
- **RSexportMethod** - funkcija domāta viena piereģistrētā datu objekta eksportam norādītajā failā.

Funkciju apraksti tiek saglabāti datu bāzē un ir pieejami visiem lietotājiem.

3.1. Funkciju parametri un atgriežamās vērtības

Visiem funkcijas apraksta objektiem ir virkne ar kopējiem laukiem:

MethodName	Bibliotēkas nosaukums (attiecīgā .DLL faila nosaukums bez ceļa un paplašinājuma)
Mfunction	Funkcijas nosaukums
TxtParameters	Speciāls tekstiskā rinda, kas satur funkcijai nododamus skalāros parametrus formātā: *<param.vārds>=<param.vērtība>;* Funkcijas aprakstā var nodefinēt noklusēto variantu šai rindai.
Comment	Komentārs: brīvā teksta funkcijas apraksts (funkcijas izsaukšanā nepiedalās)

Pārējie parametru apraksti ir specifiski katrs savam veidam:

RSimportMethod

DestRSimage	Ja uzstādīts, tad funkcijas rezultāts ir *RSimage* objekts
DestLIDARdata	Ja uzstādīts, tad funkcijas rezultāts ir *LIDARdata* objekts
Destlabels	Ja uzstādīts, tad funkcijas rezultāts ir *CategoryLabels* objekts

Vienai funkcijai drīkst uzstādīt tikai vienu no trim.

RSprocMethod

SourceRSimage1	Ja ir uzstādīts, tad jānorāda *RSimage* kā pirmais parametrs
SourceRSimage2	Ja ir uzstādīts, tad jānorāda *RSimage* kā nākamais parametrs
SourceLIDARdata	Ja ir uzstādīts, tad jānorāda *LIDARdata* kā nākamais parametrs
MaskImage	Ja ir uzstādīts, tad jānorāda *ImageMask* kā nākamais parametrs
LabelImage	Ja ir uzstādīts, tad jānorāda *CategoryLabels* kā nākamais parametrs

DestRSimage1	Ja uzstādīts, tad funkcijas rezultāts ir *RSimage* objekts
DestRSimage2	Ja uzstādīts, tad funkcijas rezultāts ir *RSimage* objekts
DestMask	Ja uzstādīts, tad funkcijas rezultāts ir *LIDARdata* objekts
DestLabels	Ja uzstādīts, tad funkcijas rezultāts ir *CategoryLabels* objekts

Vienai funkcijai drīkst uzstādīt gan vairākus parametrus, gan vairākus rezultātus. Funkcijas izsaukumā gan parametri, gan rezultāti nāks tādā pašā secībā, kā te ir uzrakstīts. Ja kāds parametrs netiks uzstādīts funkcijas aprakstā, tad to izsaukumā izlaiž vispār.

RSexportMethod

SourceRSimage	Ja ir uzstādīts, tad jānorāda *RSimage* kā parametrs
SourceLIDARdata	Ja ir uzstādīts, tad jānorāda *LIDARdata* kā parametrs
SourceLabels	Ja ir uzstādīts, tad jānorāda *CategoryLabels* kā parametrs

Vienai funkcijai drīkst uzstādīt tikai vienu no trim parametriem.

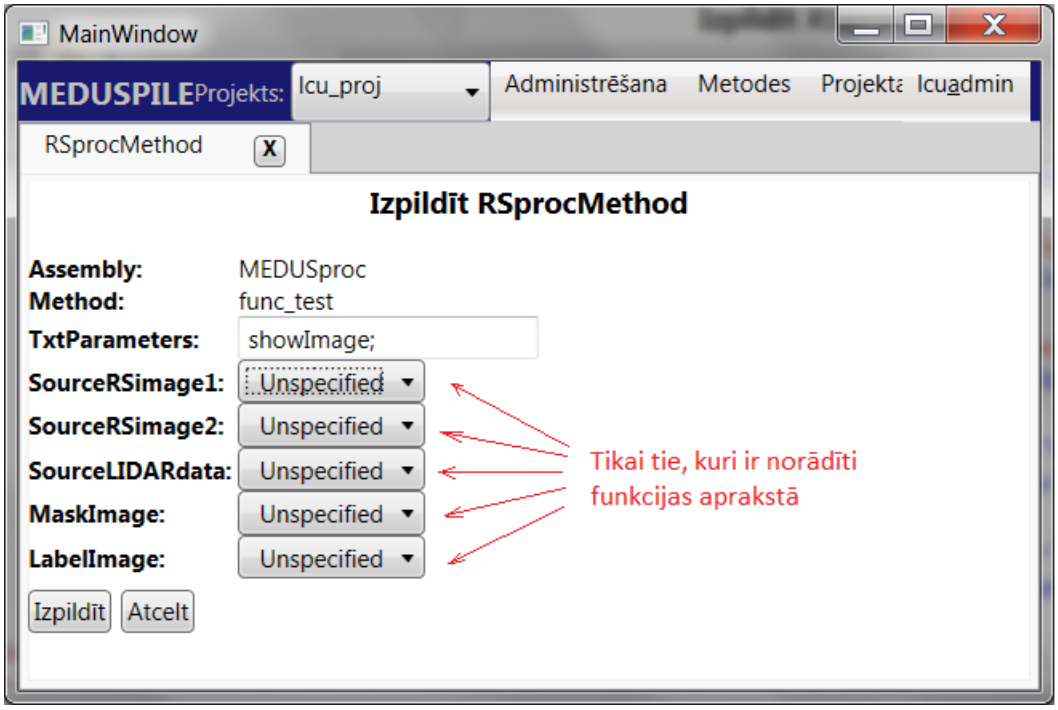
3.2. Funkciju reģistrācijas lietotāja saskarne

Funkciju reģistrācijas objektu formas ir izveidotas, lietojot noģenerētos skatus šiem objektiem. Papildus tiek pielikti nosacījumi uz parametru/rezultātu pazīmēm, lai, atbilstoši specifikācijai, nodrošinātu pārbaudes, kurā gadījumā var uzstādīt vairākas pazīmes reizē, kur - tikai vienu no iespējamajiem.

4. Datu apstrādes funkciju izsaukšana

4.1. Funkcijas izsaukšanas lietotāja saskarne

Datu apstrādes funkciju reģistrā ir iespēja arī izsaukt izvēlēto funkciju. Tiek parādīta forma, kas, atbilstoši funkcijas aprakstam, piedāvā aizpildīt datu objektu parametrus (un, ja vajag, pamainīt *TxtParameters* vērtību) un izsaukt funkciju.



Zīmējums 1. Funkcijas izsaukšanas formas piemērs (*RSprocMethod*).

Izsaukšanas forma ir manuāli izstrādāta, lai varētu dinamiski pielāgot saskarni funkcijas aprakstam.

Funkcijas izsaukšana automātiski papildina *TxtParameters* ar papildus parametru:

ProjectName=<tekošā projekta nosaukums>;

RsimportMethod/*RSexportMethod* gadījumos arī ar:

FileName=<izvēlētais fails>

4.2. Funkcijas izsaukšanas realizācija

Funkcijas izsaukšana tiek realizēta tāpat kā jebkura biznesa loģikas funkcija ar speciāli izstrādātu parametru, kas ietver sevī lietas, vajadzīgas palaišanai: norādi uz funkcijas aprakstu un lietotāja ievadītos parametrus (parametra objekts ir derīgs visiem funkciju veidiem - sk. pielikumu):

```
public void run(ML_Object item);
```

Datu apstrādes funkcijas izsaukšana notiek tā, kā ir aprakstīts paraugprojekta specifikācijā ([1], pielikumā). Šeit aprakstīsim tikai atšķirības, kas bija vajadzīgas, lai vispārinātu funkcijas izsaukšanu:

- bibliotēku pēc nosaukuma sameklē direktorija, kas ir norādīta konfigurācijas parametrā "*DLLpath*" (vērtību uzstāda katrs lietotājs, līdz ar to reālā bibliotēku atrašanas vieta varētu atšķirties katram lietotājam),
- funkcijas parametrus aizpilda no datu bāzē esošajiem objektiem, vadoties no funkcijas apraksta un lietotāju ievada *ML_Object* parametrā,
- funkcijas rezultātus nolasa, vadoties no funkcijas apraksta: datus iepilda sistēmas objektos un saglabā datu bāzē.

Piemēru funkcijas izsaukšanas realizācijai paraugprojektā var apskatīties pielikumā.

5. Rezultāti

Aktivitātes ietvaros tika izstrādāta un implementēta shēma, kā paraugprojektā var būt integrētas ārējo bibliotēku datu apstrādes funkcijas. Izstrādātā shēma ļauj lietotājiem pašiem papildināt un modificēt funkciju klāstu, nepārstrādājot paraugprojekta sistēmu.

6. Literatūras saraksts

[1] Nr. 5.1. "Prasību specifikācijas izstrāde"

7. Pielikumi

7.1. Datu apstrādes funkcijas izsaukšanas parametrs *ML_Object*

```
namespace MEDUSPILE {
  public class MLobject : DataViewObjectUniversal {
    public enum MLkind { RSimportMethod=1, RSprocMethod, RSexportMethod };

    private enum MDS_DVcolumn { kind, id, TxtParameters, fileName, projectId,
SourceRSimage1, SourceRSimage2, SourceLIDARdata, MaskImage, LabelImage,
SourceRSimage, SourceLabels};

    public int kind  { get { return this.MDSgetM_int((int)MDS_DVcolumn.kind); }
set {this.MDSsetM_int((int)MDS_DVcolumn.kind,value); OnPropertyChanged("kind"); }
}
    public int id  { get { return this.MDSgetM_int((int)MDS_DVcolumn.@id); } set
{this.MDSsetM_int((int)MDS_DVcolumn.@id,value); OnPropertyChanged("id"); } }
    public string TxtParameters  { get { return
this.MDSgetO_string((int)MDS_DVcolumn.TxtParameters); } set
{this.MDSsetM_string((int)MDS_DVcolumn.TxtParameters,value);
OnPropertyChanged("TxtParameters"); } }

    public string fileName { get { return
this.MDSgetO_string((int)MDS_DVcolumn.fileName); } set {
this.MDSsetM_string((int)MDS_DVcolumn.fileName, value);
OnPropertyChanged("fileName"); } }
    public int projectId { get { return
this.MDSgetM_int((int)MDS_DVcolumn.@projectId); } set {
this.MDSsetM_int((int)MDS_DVcolumn.@projectId, value);
OnPropertyChanged("projectId"); } }
    public int SourceRSimage1 { get { return
this.MDSgetM_int((int)MDS_DVcolumn.SourceRSimage1); } set {
this.MDSsetM_int((int)MDS_DVcolumn.SourceRSimage1, value);
OnPropertyChanged("SourceRSimage1"); } }
    public int SourceRSimage2 { get { return
this.MDSgetM_int((int)MDS_DVcolumn.SourceRSimage2); } set {
this.MDSsetM_int((int)MDS_DVcolumn.SourceRSimage2, value);
OnPropertyChanged("SourceRSimage2"); } }
    public int SourceLIDARdata { get { return
this.MDSgetM_int((int)MDS_DVcolumn.SourceLIDARdata); } set {
this.MDSsetM_int((int)MDS_DVcolumn.SourceLIDARdata, value);
OnPropertyChanged("SourceLIDARdata"); } }
    public int MaskImage { get { return
this.MDSgetM_int((int)MDS_DVcolumn.MaskImage); } set {
this.MDSsetM_int((int)MDS_DVcolumn.MaskImage, value);
OnPropertyChanged("MaskImage"); } }
    public int LabelImage { get { return
this.MDSgetM_int((int)MDS_DVcolumn.LabelImage); } set {
this.MDSsetM_int((int)MDS_DVcolumn.LabelImage, value);
OnPropertyChanged("LabelImage"); } }
    public int SourceRSimage { get { return
this.MDSgetM_int((int)MDS_DVcolumn.SourceRSimage); } set {
```

```
this.MDSsetM_int((int)MDS_DVcolumn.SourceRSimage, value);
OnPropertyChanged("SourceRSimage"); } }
    public int SourceLabels { get { return
this.MDSgetM_int((int)MDS_DVcolumn.SourceLabels); } set {
this.MDSsetM_int((int)MDS_DVcolumn.SourceLabels, value);
OnPropertyChanged("SourceLabels"); } }
  }
}
```

7.2. Datu apstrādes funkcijas izsaukšanas piemērs

Piemērs ir ar *RSimportMethod* funkcijas tipu. Te ir koda izgriezums, kas demonstrē datu apstrādes funkcijas izsaukšanu.

Bibliotēkas/funkcijas meklēšana

```
  public class FCmatlabCaller {
    public delegate void matlabFunction(int count, ref MWArray[] results,
MWArray[] inputs);

    public matlabFunction getMethod(string assemblyName, string methodName) {
      var DLLdir=ConfigurationManager.AppSettings["DLLpath"];
      var dllPath=(DLLdir==null?"":DLLdir)+assemblyName+".DLL";
      var dll=Assembly.LoadFrom(dllPath);
      var ct=string.Format("{0}.Class1", assemblyName);
      var ci=dll.CreateInstance(ct);
      var cT=ci.GetType();
      var mi=cT.GetMethods(BindingFlags.Public |
BindingFlags.Instance).Where(o=>o.Name==methodName && o.ReturnType==typeof(void)
&& o.GetParameters().Length==3).First();
      var d=Delegate.CreateDelegate(typeof(matlabFunction), ci, mi);
      var dr=d as matlabFunction;
      return dr;
     }
    }
```

Rezultāta kopēšana datu objektā (RSimage)

```
    private RSimage_N_R decode_RSimage(MWStructArray imageOut)
    {
        var item = new RSimage_N_R();
        var st_obj = new CLFileType_N_R().startSelect().Where(o => o.Name ==
"RSimage").FirstOrDefault();
        item.RSprojectData_SensorType = st_obj.id;
        item.RSprojectData_Name = imageOut.GetField("Name").ToString();
        item.RSprojectData_FileName = imageOut.GetField("Filename").ToString();
        var ft = imageOut.GetField("FileType").ToString();
        item.FileType = getFileType(ft);
        item.Samples = (int)(MWNumericArray)imageOut.GetField("samples");
        item.Lines = (int)(MWNumericArray)imageOut.GetField("lines");
        item.Bands = (int)(MWNumericArray)imageOut.GetField("bands");
        item.VisBandR = (int)(MWNumericArray)imageOut.GetField("VisBandR");
        item.VisBandG = (int)(MWNumericArray)imageOut.GetField("VisBandG");
        item.VisBandB = (int)(MWNumericArray)imageOut.GetField("VisBandB");
        var x = (double)(MWNumericArray)imageOut.GetField("PixelSizeX");
```

```
        item.PixelSizeX = Convert.ToDecimal(x);
        var y = (double)(MWNumericArray)imageOut.GetField("PixelSizeY");
        item.PixelSizeY = Convert.ToDecimal(y);
        x = (double)(MWNumericArray)imageOut.GetField("X0");
        item.X0 = Convert.ToDecimal(x);
        y = (double)(MWNumericArray)imageOut.GetField("Y0");
        item.Y0 = Convert.ToDecimal(y);
        return item;
    }
    private int getFileType(string ft)
    {
        var ft_obj = new CLFileType_N_R().startSelect().Where(o => o.Name ==
ft).FirstOrDefault();
        if (ft_obj == null)
        {
            var obj = new CLFileType_N_R();
            obj.Name = ft;
            ft_obj = obj.insert() as CLFileType_N_R;
        }
        return ft_obj.id;
    }
```

Funkcijas izsaukšana

```
...
      FCmatlabCaller.matlabFunction mp=null;
      MWArray[] res=null;
      switch ((MLobject.MLkind)(me.kind)) {
        case MLobject.MLkind.RSimportMethod:
          {
              var ir = new RSimportMethod().getByKey(new { id = me.id }) as
RSimportMethod;
              mp = new FCmatlabCaller().getMethod(ir.MethodName, ir.Mfunction);
              mp(3, ref res, new MWArray[] { (MWArray)me.TxtParameters });
              var rc = (int)(MWNumericArray)res[0];
              if (rc != 0) throw new Exception(((MWCharArray)res[2]).ToString());
              if (ir.DestRSimage) {
                  var x = decode_RSimage((MWStructArray)res[1]);
                  x.RSprojectData_ProjectId = me.projectId;
                  x.insert();
              }
              else if (ir.DestLIDARdata) {
                  ...
          }
          break;
     }
```