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. 3.7 "Biznesa objektu meta modeļa, kas kalpotu kā pamatmodelis biznesa objektiem, izstrāde" progresa pārskats

Pārskats Nr. 27 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 "Biznesa objektu meta modeļa, kas kalpotu kā pamatmodelis biznesa objektiem, izstrāde" ietvaros veikti šādi darbi:

1. Kompleksu līmeņa infrastruktūras izstrāde:
    a. Dažādu projektu tipu prasību izpēte kompleksu līmeņa infrastruktūrai;
    b. Kompleksu līmeņu interfeisa izstrāde;
    c. Savienojumu līmeņa papildināšana - kompleksu līmeņa nepieciešamās funkcionalitātes nodrošināšanai;
2. Ģenerācijas izstrāde:
    a. Ģenerācijas infrastruktūras izpēte;
    b. Ģenerācijas infrastruktūras pielāgošana kompleksu līmeņa īpatnībām dažādu projektu tipu kontekstā;
    c. Ģenerējamo daļu struktūras izstrāde;
    d. Ģenerācijas skriptu izstrāde;
3. Aktivitātes pētnieciskā darbība apspriesta ik nedēļas projekta semināros.

2. Ievads

Šī zinātniskā atskaite ir veltīta projekta apakšaktivitātes Nr.3.7 "Biznesa objektu meta modeļa, kas kalpotu kā pamatmodelis biznesa objektiem, izstrāde" ietvaros veiktajai izstrādei pārskata periodā (2015-01-01. – 2015-06-30.).

Iepriekšējā periodā šīs aktivitātes ietvaros tika izstrādāts kompleksu metamodelis, ar kura palīdzību ir iespējams aprakstīt kompleksus, kas kalpos par pamatu biznesa objektu aprakstīšanai modelī. Kā arī tika izstrādāta transformācija, kas no loģiskā modeļa pārnesa datus uz tehnoloģisko modeli.

Pašreizējais uzdevums ir, balstoties uz tehnoloģiskā modeļa, saražot kodu kompleksu nodrošināšanai izstrādājamajā sistēmā.

3. Kompleksu līmenis izstrādājamajā sistēmā

Kompleksu līmenis kalpo kā starpnieks datu plūsmā biznesa loģiskas pusē: starp savienojumiem un biznesa objektiem. Līdz ar to tam ir sekojošie uzdevumi: plakanizācija un sastrukturēšana, kā arī interfeisa nodrošināšana.

Sistēmas arhitektūrā kompleksiem ir izstrādāti kopējie mehānismi, kas realizē kompleksu līmeņa loģiku. Tomēr šie mehānismi ietver sevī daļas, kurām ir nepieciešama individualizācija, kas nāks no modeļa ar ģenerācijas palīdzību.

3.1. Plakanizācija

Plakanizācijas uzdevums ir reprezentēt datus, kas atrodas sasaistīto savienojumu kokveida struktūrā, kā viena objekta viena līmeņa atribūtus. Šis darbs prasa organizēt atribūtu nosaukumu shēmu, lai nodrošinātu, ka atribūti nepārklās viens otru. Tas tiek risināts ar to, ka atribūta nosaukuma priekšā pieliek attiecīgā savienojuma nosaukumu.

3.2. Sastrukturēšana

Sastrukturēšanas uzdevums ir atjaunot no plakanizēta objekta ar to saistīto savienojumu struktūru. Tas prasa uzturēt atbilstību starp plakanizēta objekta atribūtiem un to līdziniekiem savienojumu struktūrā. Šīs atbilstības tiek realizētas ar konvertācijas funkcijām, kas spēj iegūt datu lauka atslēgu savienojumā no kompleksa datu lauka atslēgas un otrādi.

3.3. Kompleksu līmeņa interfeiss

Būdams starpnieks, kompleksu līmenis nodrošina datu plūsmu starp savienojumiem un biznesa objektiem. Līdz ar to tas nodrošina datu operācijas biznesa objektu līmenim - atlasīšanu, filtrēšanu, izveidošanu, mainīšanu un izmešanu - izmantojot savienojumu līmeņa interfeisu.

4. Ģenerācijas infrastruktūra

Ģenerācija tiek organizēta ar Microsoft T4 transformāciju palīdzību [1]. Transformāciju izstrādei izmantojam T4 iekļaujamos failus (*.ttinclude*). Tas nodrošina, ka viens un tas pats transformācijas ķermenis var tikt izmantots vairākos projektos, bet, savukārt, pieprasa, lai viens un tas pats ķermenis būtu derīgs neatkarīgi no individuālā projekta mērķa un modeļa.

Kompleksu tehnoloģiskā modeļa lasīšanai (no modeļa faila uz atmiņas struktūru) izmantojam speciāli izstrādāto moduli, kas ir balstīts uz kompleksu tehnoloģiskā metamodeļa.

5. Kompleksu ģenerācija

5.1. Complexes.enums.ttinclude

Ģenerējamais kods nodefinē atslēgas kompleksa datu glabātuvē katram laukam plakanizētajā objektā, izmantojot pārskaitījumu (enum). Šis kods tiek izmantots izstrādājamajā sistēmā gan servera, gan klienta pusē.

5.2. Complexes.ttinclude

Ģenerējamais kods realizē individualizētās daļas visiem kompleksu līmeņa uzdevumiem: nodefinē kompleksa plakanizētos laukus, funkcijas, kas nodrošinās saites ar kompleksu laukiem un realizē kompleksu interfeisu katram kompleksam, kas ir definēts izmantojamā tehnoloģiskā modelī.

6. Rezultāti

Aktivitātes ietvaros veiktā darba rezultātā tika izstrādāta kompleksu līmeņa ģenerācijas transformācijas, kas saražo izstrādājamās sistēmas kodu ar kompleksu individuālajām daļām, balstoties uz izmantoto tehnoloģisko modeli.

7. Literatūras saraksts

[1] Code Generation and T4 Text Templates (https://msdn.microsoft.com/en-us/library/bb126445.aspx)

8. Pielikumi

8.1. Complexes.enums.ttinclude

```
<#@ include file="GenCodeComplex.ttinclude" #>
<#+ void gogo(string signature=null){
 ComplexGen(this.Host.ResolvePath("./../../../Complexes.xml"), signature);
#>
using System;
using MEDUSR;
using MEDUSP;
using System.Linq;
using System.Runtime.Serialization;
using System.Linq.Dynamic;
using System.Collections.Generic;

namespace MEDUSBL
{
<#+  var bcS = model.MDSfind_Complex();
 foreach (Complex bc in bcS){
     Dictionary<string,List<CPart>> dict = GetDBCParts(bc);
 #>
     public partial class <#=bc.name  #> : BaseComplex<<#=bc.name  #>>
        {
         public <#=bc.name  #>():base()
         {
         }
         public <#=bc.name  #>(DataContainer dc):base(dc)
         {
         }

<#+
foreach (CPart  cp in bc.CPartS ){
 string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;
#>
         // CPart "<#= cp.name #>"
         public <#= joinNamespace #>.<#= cp.joinName #> MDSJoin_<#= cp.name #>  {
set; get; }
         protected IQueryable<<#= bc.name #>> <#=cp.name #>;
<#+
  foreach (JPart  jp in cp.JPartS ){
#>
         // JPart <#= jp.name #>
<#+    foreach (CField  cf in jp.CFieldS )
   {
          string pr = GenProperty(cf);
                #>
               <#= pr #>
 <#+
    }
       }
}
#>
```

```
        override public IQueryable select(IStoreContext tc, SelectionDescriptor
sd=null)
        {
                    MDSbeforeSelect();
              Filter f = null;
               if (sd != null && sd.filter != null)
               {
                     //create filter
                     f = new Filter(sd.filter);
               }

<#+
//one cpart
var tolist = (bc.CPartS.Count==1)?"":".ToList().AsQueryable()";
if (bc.CPartS.Count==1 ){
#>
              <#= bc.CPartS[0].name #> =  _sel_<#= bc.CPartS[0].name #>(tc);
                        <#= bc.CPartS[0].name #> = mds_procesDescriptor(<#=
bc.CPartS[0].name #>, sd) as IQueryable<<#= bc.name #>>;
              var ret = <#=  bc.CPartS[0].name#>;
<#+
//one cpart end
}
else{
//many cparts
#>
              Dictionary<string,Filter> dict = null;
                       if (f != null)
                            dict = this.mds_splitFilter(f);
<#+
foreach (CPart  cp in bc.CPartS ){
 string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;
#>
              var q_<#= cp.name #>  = (new <#= joinNamespace #>.<#= cp.joinName
#>()).select(tc)  as IQueryable<<#= joinNamespace #>.<#= cp.joinName #>>;
              q_<#= cp.name #>  = mds_getPartQueryable<<#= joinNamespace #>.<#=
cp.joinName #>>(tc,q_<#= cp.name #>, dict, "<#= cp.name #>");
              q_<#= cp.name #> = q_<#= cp.name #><#=tolist#>;
<#+
  } //foreach end
#>
                        var ret =
<#+
foreach (CPart  cp in bc.CPartS ){
#>
                                        from <#= cp.name #> in q_<#= cp.name #>
<#+
  } //foreach end
#>
<#+
  List <string>  sl = new List<string>();
foreach (CPart  cp in bc.CPartS ){
   // dictionary fieldname/JPart.name
   Dictionary<string,string> fieldsJPn = new Dictionary<string,string>();
  foreach (JPart  jp in cp.JPartS ){
```

```
   foreach (CField  cf in jp.CFieldS ){
      if (fieldsJPn.ContainsKey(cf.fieldName))
                fieldsJPn.Add(" !!! error duplicate fieldName " +
cf.fieldName,jp.name +" !!! error duplicate fieldName "+ cf.fieldName);
      else
                fieldsJPn.Add(cf.fieldName,jp.name);
   }
  }
  sl.Add(CreateProps(cp,fieldsJPn,",",cp.name));

  } //foreach end
#>
                        select new <#= bc.name #>
                        {
<#=  string.Join(","+Environment.NewLine,sl) #>
                        };

             ret  = mds_getPartQueryable<<#= bc.name #>>(tc,ret, dict, "_") as
IQueryable<<#= bc.name #>>;
                       ret = mds_procesDescriptor(ret, sd) as IQueryable<<#= bc.name
#>>;

<#+
}
//many cparts end
#>
                       MDSafterSelect();
                       return ret;
        }

        public override IQueryable  reselect(IStoreContext tc,Type viewEnum=null)
        {
<#+
int j =1;
foreach (CPart  cp in bc.CPartS ){
#>
            var j<#= j.ToString() #> = _sel_<#= cp.name #>(tc,
viewEnum)<#=tolist#>;
<#+
       j++;
}
#>
            return j1; //new <#=bc.name  #>(r1);
        }

        public override BaseComplex reselectComplex(IStoreContext tc, Type
viewEnum = null)
        {
            return (reselect(tc, viewEnum) as IQueryable<<#=bc.name
#>>).FirstOrDefault();
        }


        public IQueryable<<#=bc.name  #>> selectNeighbor(IStoreContext
tc,DataViewObject me = null,int? via = null,bool isReverse = false)
             {
<#+
```

```
int k =1;
foreach (CPart  cp in bc.CPartS ){
#>
            var j<#= k.ToString() #> = _sel_<#= cp.name #>(tc,
null,me,via,isReverse);
<#+
       k++;
}
#>
            return j1;
              }

        public override object getKey(Type viewEnum = null)
        {
            createJoins(viewEnum);
            return _myJoins[0].getKey();
        }

        override public string[] getOrderByClause()
        {
            IList<string> l = new List<string>();
<#+
k =1;
foreach (CPart  cp in bc.CPartS ){
        string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;
#>
            var j<#= k.ToString() #> = new <#= joinNamespace #>.<#= cp.joinName
#>();
            var oc<#= k.ToString() #> = j<#= k.ToString() #>.getOrderByClause();
            foreach (string s in oc<#= k.ToString() #>)
            {
                l.Add(mds_joinFieldToComplexField(s));
            }
<#+
       k++;
}
#>
                  return l.ToArray();
              }

        protected override string mds_joinNameFromCPartName(string cpartName)
        {
                  string retVal = null;
                  switch (cpartName)
                     {
<#+
foreach (CPart  cp in bc.CPartS ){
#>
               case "<#= cp.name #>":
                           retVal = "<#= cp.joinName #>";
                        break;
<#+
                }
#>
                     }
            return retVal;
```

```
            }

        protected override string mds_complexFieldToJoinField(string  s)
        {
                  string retVal = null;
                  switch (s)
                     {
<#+
foreach (CPart  cp in bc.CPartS ){
       foreach (JPart  jp in cp.JPartS ){
                     foreach (CField  cf in jp.CFieldS ){
                            if (cf.isRefName != true)
                {
#>
               case "<#= cf.fieldName #>":
                           retVal = "<#= jp.name #>.<#= cf.name #>";
                        break;
<#+
                }
            }
    }
}
#>
                     }
            return retVal;
        }

        protected override string mds_getCPartName(string  s)
        {
<#+
        if (bc.CPartS.Count==1)
              {
#>
            return "<#= bc.CPartS[0].name #>";
<#+
} else{
#>
                  string retVal = null;
                  switch (s)
                     {
<#+
foreach (CPart  cp in bc.CPartS ){
       foreach (JPart  jp in cp.JPartS ){
                     foreach (CField  cf in jp.CFieldS ){
                            if (cf.isRefName != true)
                {
#>
               case "<#= cf.fieldName #>":
<#+
                }
            }
    }
#>
                        retVal = "<#= cp.name #>";
                     break;
<#+
}
```

```
#>
                    }
            return retVal;
<#+
}
var cmnt = (bc.CPartS.Count == 1)?"":"//";
#>
        }

        private string mds_joinFieldToComplexField(string  s)
        {
                string retVal = s;
       <#= cmnt #>       switch (s)
       <#= cmnt #>          {
<#+

foreach (CPart  cp in bc.CPartS ){
       foreach (JPart  jp in cp.JPartS ){
                    foreach (CField  cf in jp.CFieldS ){
                           if (cf.isRefName != true)
               {
#>
             <#= cmnt #>  case "<#= jp.name #>.<#= cf.name #>":
                      <#= cmnt #>    retVal = "<#= cf.fieldName #>";
                      <#= cmnt #>  break;
<#+
               }
            }
    }
}
#>
       <#= cmnt #>          }
            return retVal;
        }
        protected override BaseComplex updateJoins(IStoreContext tc, Type
viewEnum)
        {
<#+
foreach (CPart  cp in bc.CPartS ){
 string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;
 string db = string.IsNullOrEmpty(cp.dataBase)?signature:cp.dataBase;
#>
            {
            <#= getContextCheck(db) #>
                    var  o = MDSJoin_<#= cp.name #>;
<#= InitJoin(bc,cp)  #>
            MDSJoin_<#= cp.name #>.update(tc);
                    }
<#+                 } #>
           return this;
        }

        protected override void assignProperties(JoinObject j, Type viewEnum,int
cp)
        {
                MDSComplexCpart cpe = (MDSComplexCpart) cp;
```

```
                switch (cpe)
                {
<#+
foreach (CPart  cp in bc.CPartS ){
string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;
#>
           case MDSComplexCpart.<#=cp.name #>:
               assignProps_<#= cp.name#>(j,viewEnum);
           break;
<#+
}
#>
           }
        }

        protected override void createJoins(Type viewEnum= null)
        {
              _myJoins = new JoinObject[<#=bc.CPartS.Count.ToString()  #>];
<#+
foreach (CPart  cp in bc.CPartS ){
string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;
#>
             createJoin_<#= cp.name #>(viewEnum);
<#+
}
#>
        }

<#+
int l = 0;
foreach (CPart  cp in bc.CPartS ){
string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;

#>

        private <#= joinNamespace #>.<#= cp.joinName #> createJoin_<#= cp.name
#>(Type viewEnum= null)
        {
                  if (_myJoins==null)
                        _myJoins = new JoinObject[<#=bc.CPartS.Count.ToString()
#>];
                  var o = new <#= joinNamespace #>.<#= cp.joinName #>();
                     MDSJoin_<#= cp.name #> = o;
                     _myJoins[<#= l.ToString() #>] = o;
<#= InitJoin(bc,cp)  #>
                return o;
        }
<#+
l++;
}
#>
        private object returnFK(int? via)
        {
```

```
<#= findFK(bc)  #>
            return null;
        }



<#+
        foreach ( KeyValuePair<string,List<CPart>> kv in dict)
     {
        foreach ( CPart cp in kv.Value)
     {
   // dictionary fieldname/JPart.name
   Dictionary<string,string> fieldsJPn = new Dictionary<string,string>();
  foreach (JPart  jp in cp.JPartS ){
   foreach (CField  cf in jp.CFieldS ){
      if (fieldsJPn.ContainsKey(cf.fieldName))
               fieldsJPn.Add(" !!! error duplicate fieldName " +
cf.fieldName,jp.name +" !!! error duplicate fieldName "+ cf.fieldName);
      else
               fieldsJPn.Add(cf.fieldName,jp.name);
   }
  }
         string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;
         string db = string.IsNullOrEmpty(cp.dataBase)?signature:cp.dataBase;
#>
        private IQueryable<<#= bc.name #>> _sel_<#= cp.name  #>(IStoreContext
tc,Type viewEnum=null,DataViewObject me = null,int? via = null,bool isReverse =
false)
        {
            // DB specific context
            <#= getContextCheck(db) #>
                  IQueryable<<#= joinNamespace #>.<#= cp.joinName #>> j;
            if (viewEnum==null)
                    {
                if (me==null || isReverse)
                {
                   j = (new <#= joinNamespace #>.<#= cp.joinName #>()).select(tc)
as IQueryable<<#= joinNamespace #>.<#= cp.joinName #>>;
                }
                else
                           {
                   var cmpl =
Activator.CreateInstance(me.getComplexType(),me.MDSextract);
                   object fk = returnFK(via);
                   j = (new <#= joinNamespace #>.<#= cp.joinName #>())._sel(tc,
fk);
                           }
            }
            else
                     {
               j = createJoin_<#= cp.name  #>(viewEnum).reselect(tc);
            }

            <#= cp.name #>  =  j.Select(o =>
              new <#= bc.name #>() {
<#= CreateProps(cp,fieldsJPn,",")  #>
```

```
                        }) as IQueryable<<#= bc.name #>>;


            if (isReverse)
            {
                object pk = me.getKey();
                // only for simple key!!!
                            // only for 1 CPart!!!
                var pifs = pk.GetType().GetProperties();
                string condition = string.Format("{0}={1}",
Enum.GetName(typeof(MDSComplexField), via), pifs[0].GetValue(pk).ToString());
                <#=bc.CPartS[0].name#> = <#=bc.CPartS[0].name#>.Where(condition);
            }
        return <#=bc.CPartS[0].name#>;
        }

        private  void assignProps_<#= cp.name#>(JoinObject j, Type viewEnum)
        {
                <#= joinNamespace #>.<#= cp.joinName #> o = j as  <#= joinNamespace
#>.<#= cp.joinName #>;
<#= CreateProps(cp,fieldsJPn,";")  #>;
        }

 <#+
    }
}
#>
    } //class
<#+ } #>
} // namespace
<#+ } #>

<#+
 ////////////////////////////////////////////////////////////////////////////
        string GenProperty(CField cf)
        {
           string rez = "";
           string mandOpt = (cf.isMandatoryInJoin == true)?"M":"O";
           string ctype = MEDUS.TypeConvertor.ToNetType(cf.type);
           if (ctype == "byte[]")
               ctype = "byteA";
             string  defaultValue = "";

               string realType = (cf.isMandatoryInJoin ==
true)?MEDUS.TypeConvertor.ToNetType(cf.type):MEDUS.TypeConvertor.ToNetNullableType
(cf.type);

           rez += string.Format("public {0} @{1}  {{ get {{ return
this.MDSget{2}_{3}((int)MDSComplexField.@{1}{4}); }} set
{{this.MDSset{2}_{3}((int)MDSComplexField.@{1},value); }} }}", realType,
cf.fieldName, mandOpt, ctype,defaultValue);

           return rez;
        }

  string CreateProps(CPart cp,Dictionary<string,string> d,string separator,string
varName="o")
```

```
{
       List<string> l = new List<string>();
    int i = 0;
              foreach ( JPart jp in cp.JPartS)
        {
                      foreach (CField cf in jp.CFieldS)
             {
                        string cName = cf.name;
                string jpName = jp.name;

                            if (cf.isRefName == true)
                {
                                if (jp.refName == null)
                                          cName = "NavRefNames";
                    else
                    {
                                       cName = jp.refName.name;
                                    d.TryGetValue(jp.refName.fieldName,out jpName);
                    }
                }


                            string right =
string.Format("{0}.@{1}.@{2}",varName,jpName,cName);
                            string ifCondition = "";
                            if (separator == ";")
                                ifCondition = string.Format("       if (viewEnum ==
null || Enum.IsDefined(viewEnum,\"{0}\"))", cf.fieldName);


                            if (cf.isRefName == true)
                {
                                    if( jp.refName != null)
                    {
                                              string convFunc = "";
                                                 if (jp.refName.type != "String" &&
jp.refName.type != "string")
                                                      convFunc = ".ToString()";

                                     right = string.Format("{0}{1}","@"+right,convFunc);
                                    //            break;
                    //  }
                    }
                                    else
                    {
                                           right = "\"Nav refNames\"";
                    }
                  l.Add(string.Format("           {2} @{0} =
{1}",cf.fieldName,right,ifCondition));

                }
                            else
                            {
                  l.Add(string.Format("           {2} @{0} =
{1}",cf.fieldName,right,ifCondition));
                            }
                          i++;
```

```
            }
         }
        return string.Join(separator+Environment.NewLine,l);
 }


 string InitJoin(Complex bc,CPart cp)
 {
        List<string> l = new List<string>();
               foreach ( JPart jp in cp.JPartS)
         {
                        foreach (CField cf in jp.CFieldS)
              {
                          string cName = cf.name;
                string jpName = jp.name;

                              string left =
string.Format("o.@{0}.@{1}",jpName,cName);
                              if (cf.isRefName != true)
                {
                                   string right = "@"+cf.fieldName;
                                      if (cf.isMandatoryInTable !=
cf.isMandatoryInJoin)
                     {
                                                left = "if ("+  right  + " != null) " +
left;
                                                right = "(" +
MEDUS.TypeConvertor.ToNetType(cf.type) + ")" +right;
                                     }
                    l.Add (string.Format("         if (viewEnum == null ||
Enum.IsDefined(viewEnum,\"{0}\"))",cf.fieldName));
                          l.Add(string.Format("             {0} = {1};",left,right));
                }
              }
         }
        return string.Join(Environment.NewLine,l);
 }


 string findFK(Complex bc)
 {
        List<string> l = new List<string>();
       foreach ( CPart cp in bc.CPartS)
        {
               foreach ( JPart jp in cp.JPartS)
         {
                        foreach (CField cf in jp.CFieldS)
              {
                          string cName = cf.name;
                string jpName = jp.name;

                              if (cf.isRefName != true)
                {
                    l.Add (string.Format("         if
(((int){0}.MDSComplexField.@{1}) == via)",bc.name,cf.fieldName));
                          l.Add(string.Format("             return
@{0};",cf.fieldName));
```

```
                }
            }
        }
       }
       return string.Join(Environment.NewLine,l);
 }

#>
```

8.2. Complexes.ttinclude

```
<#@ include file="GenCodeComplex.ttinclude" #>
<#+ void gogo(string signature=null){
 ComplexGen(this.Host.ResolvePath("./../../../Complexes.xml"), signature);
#>
using System;
using MEDUSR;
using MEDUSP;
using System.Linq;
using System.Runtime.Serialization;
using System.Linq.Dynamic;
using System.Collections.Generic;

namespace MEDUSBL
{
<#+  var bcS = model.MDSfind_Complex();
 foreach (Complex bc in bcS){
     Dictionary<string,List<CPart>> dict = GetDBCParts(bc);
 #>
     public partial class <#=bc.name  #> : BaseComplex<<#=bc.name  #>>
        {
         public <#=bc.name  #>():base()
         {
         }
         public <#=bc.name  #>(DataContainer dc):base(dc)
         {
         }

<#+
foreach (CPart  cp in bc.CPartS ){
 string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;
#>
         public <#= joinNamespace #>.<#= cp.joinName #> MDSJoin_<#= cp.name #>  {
set; get; }
         protected IQueryable<<#= bc.name #>> <#=cp.name #>;
<#+
  foreach (JPart  jp in cp.JPartS ){
#>
<#+    foreach (CField  cf in jp.CFieldS )
   {
          string pr = GenProperty(cf);
               #>
              <#= pr #>
 <#+
    }
       }
```

```
}
#>

        override public IQueryable select(IStoreContext tc, SelectionDescriptor
sd=null)
        {
                    MDSbeforeSelect();
              Filter f = null;
               if (sd != null && sd.filter != null)
               {
                     //create filter
                     f = new Filter(sd.filter);
               }

<#+
//one cpart
var tolist = (bc.CPartS.Count==1)?"":".ToList().AsQueryable()";
if (bc.CPartS.Count==1 ){
#>
              <#= bc.CPartS[0].name #> =  _sel_<#= bc.CPartS[0].name #>(tc);
                        <#= bc.CPartS[0].name #> = mds_procesDescriptor(<#=
bc.CPartS[0].name #>, sd) as IQueryable<<#= bc.name #>>;
              var ret = <#=  bc.CPartS[0].name#>;
<#+
//one cpart end
}
else{
//many cparts
#>
              Dictionary<string,Filter> dict = null;
                       if (f != null)
                            dict = this.mds_splitFilter(f);
<#+
foreach (CPart  cp in bc.CPartS ){
 string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;
#>
              var q_<#= cp.name #>  = (new <#= joinNamespace #>.<#= cp.joinName
#>()).select(tc)  as IQueryable<<#= joinNamespace #>.<#= cp.joinName #>>;
              q_<#= cp.name #>  = mds_getPartQueryable<<#= joinNamespace #>.<#=
cp.joinName #>>(tc,q_<#= cp.name #>, dict, "<#= cp.name #>");
              q_<#= cp.name #> = q_<#= cp.name #><#=tolist#>;
<#+
  } //foreach end
#>
                        var ret =
<#+
foreach (CPart  cp in bc.CPartS ){
#>
                                        from <#= cp.name #> in q_<#= cp.name #>
<#+
  } //foreach end
#>
<#+
  List <string>  sl = new List<string>();
foreach (CPart  cp in bc.CPartS ){
   // dictionary fieldname/JPart.name
```

```
   Dictionary<string,string> fieldsJPn = new Dictionary<string,string>();
  foreach (JPart  jp in cp.JPartS ){
   foreach (CField  cf in jp.CFieldS ){
      if (fieldsJPn.ContainsKey(cf.fieldName))
                fieldsJPn.Add(" !!! error duplicate fieldName " +
cf.fieldName,jp.name +" !!! error duplicate fieldName "+ cf.fieldName);
      else
                fieldsJPn.Add(cf.fieldName,jp.name);
   }
  }
  sl.Add(CreateProps(cp,fieldsJPn,",",cp.name));

  } //foreach end
#>
                        select new <#= bc.name #>
                        {
<#=  string.Join(","+Environment.NewLine,sl) #>
                        };

             ret  = mds_getPartQueryable<<#= bc.name #>>(tc,ret, dict, "_") as
IQueryable<<#= bc.name #>>;
                      ret = mds_procesDescriptor(ret, sd) as IQueryable<<#= bc.name
#>>;

<#+
}
//many cparts end
#>
                      MDSafterSelect();
                      return ret;
        }

        public override IQueryable  reselect(IStoreContext tc,Type viewEnum=null)
        {
<#+
int j =1;
foreach (CPart  cp in bc.CPartS ){
#>
            var j<#= j.ToString() #> = _sel_<#= cp.name #>(tc,
viewEnum)<#=tolist#>;
<#+
       j++;
}
#>
            return j1; //new <#=bc.name  #>(r1);
        }

        public override BaseComplex reselectComplex(IStoreContext tc, Type
viewEnum = null)
        {
            return (reselect(tc, viewEnum) as IQueryable<<#=bc.name
#>>).FirstOrDefault();
        }


        public IQueryable<<#=bc.name  #>> selectNeighbor(IStoreContext
tc,DataViewObject me = null,int? via = null,bool isReverse = false)
```

```
            {
<#+
int k =1;
foreach (CPart  cp in bc.CPartS ){
#>
            var j<#= k.ToString() #> = _sel_<#= cp.name #>(tc,
null,me,via,isReverse);
<#+
       k++;
}
#>
            return j1;
              }

        public override object getKey(Type viewEnum = null)
        {
            createJoins(viewEnum);
            return _myJoins[0].getKey();
        }

        override public string[] getOrderByClause()
        {
            IList<string> l = new List<string>();
<#+
k =1;
foreach (CPart  cp in bc.CPartS ){
        string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;
#>
            var j<#= k.ToString() #> = new <#= joinNamespace #>.<#= cp.joinName
#>();
            var oc<#= k.ToString() #> = j<#= k.ToString() #>.getOrderByClause();
            foreach (string s in oc<#= k.ToString() #>)
            {
                l.Add(mds_joinFieldToComplexField(s));
            }
<#+
       k++;
}
#>
                return l.ToArray();
              }

        protected override string mds_joinNameFromCPartName(string cpartName)
        {
                string retVal = null;
                switch (cpartName)
                   {
<#+
foreach (CPart  cp in bc.CPartS ){
#>
               case "<#= cp.name #>":
                          retVal = "<#= cp.joinName #>";
                      break;
<#+
                }
#>
```

```
                    }
            return retVal;
              }

        protected override string mds_complexFieldToJoinField(string  s)
        {
                  string retVal = null;
                  switch (s)
                     {
<#+
foreach (CPart  cp in bc.CPartS ){
       foreach (JPart  jp in cp.JPartS ){
                     foreach (CField  cf in jp.CFieldS ){
                            if (cf.isRefName != true)
                {
#>
               case "<#= cf.fieldName #>":
                           retVal = "<#= jp.name #>.<#= cf.name #>";
                        break;
<#+
                }
            }
    }
}
#>
                    }
            return retVal;
        }

        protected override string mds_getCPartName(string  s)
        {
<#+
        if (bc.CPartS.Count==1)
              {
#>
            return "<#= bc.CPartS[0].name #>";
<#+
} else{
#>
                  string retVal = null;
                  switch (s)
                     {
<#+
foreach (CPart  cp in bc.CPartS ){
       foreach (JPart  jp in cp.JPartS ){
                     foreach (CField  cf in jp.CFieldS ){
                            if (cf.isRefName != true)
                {
#>
               case "<#= cf.fieldName #>":
<#+
                }
            }
    }
#>
                        retVal = "<#= cp.name #>";
                     break;
```

```
<#+
}
#>
                    }
            return retVal;
<#+
}
var cmnt = (bc.CPartS.Count == 1)?"":"//";
#>
        }

        private string mds_joinFieldToComplexField(string  s)
        {
                string retVal = s;
       <#= cmnt #>       switch (s)
       <#= cmnt #>          {
<#+

foreach (CPart  cp in bc.CPartS ){
       foreach (JPart  jp in cp.JPartS ){
                    foreach (CField  cf in jp.CFieldS ){
                           if (cf.isRefName != true)
               {
#>
             <#= cmnt #>  case "<#= jp.name #>.<#= cf.name #>":
                       <#= cmnt #>    retVal = "<#= cf.fieldName #>";
                       <#= cmnt #>  break;
<#+
               }
            }
    }
}
#>
       <#= cmnt #>          }
            return retVal;
        }
        protected override BaseComplex updateJoins(IStoreContext tc, Type
viewEnum)
        {
<#+
foreach (CPart  cp in bc.CPartS ){
 string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;
 string db = string.IsNullOrEmpty(cp.dataBase)?signature:cp.dataBase;
#>
            {
            <#= getContextCheck(db) #>
                    var  o = MDSJoin_<#= cp.name #>;
<#= InitJoin(bc,cp)  #>
            MDSJoin_<#= cp.name #>.update(tc);
                    }
<#+                 } #>
           return this;
        }

        protected override void assignProperties(JoinObject j, Type viewEnum,int
cp)
```

```
        {
                MDSComplexCpart cpe = (MDSComplexCpart) cp;
                switch (cpe)
                {
<#+
foreach (CPart  cp in bc.CPartS ){
string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;
#>
           case MDSComplexCpart.<#=cp.name #>:
               assignProps_<#= cp.name#>(j,viewEnum);
           break;
<#+
}
#>
           }
        }


        protected override void createJoins(Type viewEnum= null)
        {
              _myJoins = new JoinObject[<#=bc.CPartS.Count.ToString()  #>];
<#+
foreach (CPart  cp in bc.CPartS ){
string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;
#>
             createJoin_<#= cp.name #>(viewEnum);
<#+
}
#>
        }

<#+
int l = 0;
foreach (CPart  cp in bc.CPartS ){
string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;

#>

        private <#= joinNamespace #>.<#= cp.joinName #> createJoin_<#= cp.name
#>(Type viewEnum= null)
        {
                  if (_myJoins==null)
                        _myJoins = new JoinObject[<#=bc.CPartS.Count.ToString()
#>];
                  var o = new <#= joinNamespace #>.<#= cp.joinName #>();
                     MDSJoin_<#= cp.name #> = o;
                     _myJoins[<#= l.ToString() #>] = o;
<#= InitJoin(bc,cp)  #>
                return o;
        }
<#+
l++;
}
#>
```

```
        private object returnFK(int? via)
        {
<#= findFK(bc)  #>
              return null;
        }



<#+
        foreach ( KeyValuePair<string,List<CPart>> kv in dict)
     {
        foreach ( CPart cp in kv.Value)
     {
   // dictionary fieldname/JPart.name
   Dictionary<string,string> fieldsJPn = new Dictionary<string,string>();
  foreach (JPart  jp in cp.JPartS ){
   foreach (CField  cf in jp.CFieldS ){
      if (fieldsJPn.ContainsKey(cf.fieldName))
                fieldsJPn.Add(" !!! error duplicate fieldName " +
cf.fieldName,jp.name +" !!! error duplicate fieldName "+ cf.fieldName);
      else
                fieldsJPn.Add(cf.fieldName,jp.name);
   }
  }
         string joinNamespace = (cp.dataBase ==
null)?signature:CONSTnamespacePrefix+cp.dataBase;
         string db = string.IsNullOrEmpty(cp.dataBase)?signature:cp.dataBase;
#>
        private IQueryable<<#= bc.name #>> _sel_<#= cp.name  #>(IStoreContext
tc,Type viewEnum=null,DataViewObject me = null,int? via = null,bool isReverse =
false)
        {
            // DB specific context
            <#= getContextCheck(db) #>
                  IQueryable<<#= joinNamespace #>.<#= cp.joinName #>> j;
            if (viewEnum==null)
                     {
                if (me==null || isReverse)
                {
                   j = (new <#= joinNamespace #>.<#= cp.joinName #>()).select(tc)
as IQueryable<<#= joinNamespace #>.<#= cp.joinName #>>;
                }
                else
                           {
                   var cmpl =
Activator.CreateInstance(me.getComplexType(),me.MDSextract);
                   object fk = returnFK(via);
                   j = (new <#= joinNamespace #>.<#= cp.joinName #>())._sel(tc,
fk);
                           }
            }
            else
                     {
               j = createJoin_<#= cp.name  #>(viewEnum).reselect(tc);
            }

            <#= cp.name #>  =  j.Select(o =>
```

```
                new <#= bc.name #>() {
<#= CreateProps(cp,fieldsJPn,",")  #>
                        }) as IQueryable<<#= bc.name #>>;


            if (isReverse)
            {
                object pk = me.getKey();
                // only for simple key!!!
                            // only for 1 CPart!!!
                var pifs = pk.GetType().GetProperties();
                string condition = string.Format("{0}={1}",
Enum.GetName(typeof(MDSComplexField), via), pifs[0].GetValue(pk).ToString());
                <#=bc.CPartS[0].name#> = <#=bc.CPartS[0].name#>.Where(condition);
            }
        return <#=bc.CPartS[0].name#>;
        }

        private  void assignProps_<#= cp.name#>(JoinObject j, Type viewEnum)
        {
                <#= joinNamespace #>.<#= cp.joinName #> o = j as  <#= joinNamespace
#>.<#= cp.joinName #>;
<#= CreateProps(cp,fieldsJPn,";")  #>;
        }

 <#+
    }
}
#>
    } //class
<#+ } #>
} // namespace
<#+ } #>

<#+
 ////////////////////////////////////////////////////////////////////////////
        string GenProperty(CField cf)
        {
           string rez = "";
           string mandOpt = (cf.isMandatoryInJoin == true)?"M":"O";
           string ctype = MEDUS.TypeConvertor.ToNetType(cf.type);
           if (ctype == "byte[]")
               ctype = "byteA";
             string  defaultValue = "";

               string realType = (cf.isMandatoryInJoin ==
true)?MEDUS.TypeConvertor.ToNetType(cf.type):MEDUS.TypeConvertor.ToNetNullableType
(cf.type);

           rez += string.Format("public {0} @{1}  {{ get {{ return
this.MDSget{2}_{3}((int)MDSComplexField.@{1}{4}); }} set
{{this.MDSset{2}_{3}((int)MDSComplexField.@{1},value); }} }}", realType,
cf.fieldName, mandOpt, ctype,defaultValue);

           return rez;
        }
```

```
  string CreateProps(CPart cp,Dictionary<string,string> d,string separator,string
varName="o")
 {
        List<string> l = new List<string>();
     int i = 0;
              foreach ( JPart jp in cp.JPartS)
         {
                      foreach (CField cf in jp.CFieldS)
              {
                        string cName = cf.name;
                string jpName = jp.name;

                            if (cf.isRefName == true)
                {
                                if (jp.refName == null)
                                          cName = "NavRefNames";
                    else
                    {
                                       cName = jp.refName.name;
                                    d.TryGetValue(jp.refName.fieldName,out jpName);
                    }
                }


                            string right =
string.Format("{0}.@{1}.@{2}",varName,jpName,cName);
                            string ifCondition = "";
                            if (separator == ";")
                                ifCondition = string.Format("       if (viewEnum ==
null || Enum.IsDefined(viewEnum,\"{0}\"))", cf.fieldName);


                            if (cf.isRefName == true)
                {
                                    if( jp.refName != null)
                    {
                                               string convFunc = "";
                                                  if (jp.refName.type != "String" &&
jp.refName.type != "string")
                                                       convFunc = ".ToString()";

                                right = string.Format("{0}{1}","@"+right,convFunc);
                    }
                                    else
                    {
                                           right = "\"Nav refNames\"";
                    }
                  l.Add(string.Format("           {2} @{0} =
{1}",cf.fieldName,right,ifCondition));

                }
                            else
                            {
                  l.Add(string.Format("           {2} @{0} =
{1}",cf.fieldName,right,ifCondition));
                            }
                        i++;
```

```
            }
        }
       return string.Join(separator+Environment.NewLine,l);
}


string InitJoin(Complex bc,CPart cp)
{
       List<string> l = new List<string>();
              foreach ( JPart jp in cp.JPartS)
        {
                     foreach (CField cf in jp.CFieldS)
            {
                       string cName = cf.name;
              string jpName = jp.name;

                           string left =
string.Format("o.@{0}.@{1}",jpName,cName);
                           if (cf.isRefName != true)
              {
                               string right = "@"+cf.fieldName;
                                  if (cf.isMandatoryInTable !=
cf.isMandatoryInJoin)
                    {
                                           left = "if ("+  right  + " != null) " +
left;
                                           right = "(" +
MEDUS.TypeConvertor.ToNetType(cf.type) + ")" +right;
                                    }
                  l.Add (string.Format("         if (viewEnum == null ||
Enum.IsDefined(viewEnum,\"{0}\"))",cf.fieldName));
                      l.Add(string.Format("             {0} = {1};",left,right));
              }

            }
        }
       return string.Join(Environment.NewLine,l);
}

string findFK(Complex bc)
{
       List<string> l = new List<string>();
      foreach ( CPart cp in bc.CPartS)
       {
              foreach ( JPart jp in cp.JPartS)
        {
                     foreach (CField cf in jp.CFieldS)
            {
                       string cName = cf.name;
              string jpName = jp.name;

                           if (cf.isRefName != true)
              {
                  l.Add (string.Format("         if
(((int){0}.MDSComplexField.@{1}) == via)",bc.name,cf.fieldName));
                      l.Add(string.Format("             return
@{0};",cf.fieldName));
```

```
                }
            }
        }
    }
    return string.Join(Environment.NewLine,l);
}

#>
```