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.2.3.3 "Modeļa eksporta izveide" progresa pārskats

Pārskats Nr.18. par periodu no 2014.gada 1.jūlija līdz 2014.gada 31.decembrim.

SATURS

1. Kopsavilkums

Pārskata periodā (2014-07-01 – 2014-12-31.) projekta „Multi - modeļu izstrādes tehnoloģija .NET pielietojumu projektiem" aktivitātes Nr.2.3.3 "Modeļa eksporta izveide" ietvaros veikti šādi darbi:

1. Modeļa eksporta prasību analīze
2. Eksporta no universālās glabātuves uz modeļu apmaiņas formātu izstrāde.
3. Transformāciju valodā veidota eksporta izstrāde.
4. T4 ģenerācijas izstrāde.
5. Specifiska eksporta izpēte.
6. Aktivitātes pētnieciskā darbība apspriesta ik nedēļas projekta semināros.

2. Ievads

Šis pārskats ir veltīts projekta apakšaktivitātes Nr.2.3.3 "Modeļa eksporta izveide" ietvaros paveiktajam. Apakšaktivitātes ietvaros tika izstrādāts modeļu eksportētājs.

Pārskatā aprakstīts modeļu eksports – no universālās glabātuves uz modeļu apmaiņas formātu, transformāciju valodā veidots eksports, T4 ģenerācija un pielietojuma specifisks eksports. Pielikumā pievienots T4 bāzēta eksporta piemērs.

3. Modeļa eksports.

M2T transformācija no modeļa uz tekstu (vai kādu citu veidu) tikai vienkāršākos gadījumos varētu tikt nosaukta par eksportu. Saturīgi lielākajā daļā gadījumu būtu nepieciešama funkcionalitāte, ko parasti nodrošina atskaišu ģeneratori. Bez tam paliek situācijas, kad, saturīgi izmantojot modeļa datus, tiek veidots pilnīgi jauns rezultāts. Tāpēc nav iespējams automātisks mehānisms, kas risinātu visus dzīves gadījumus.

Lielā mērā eksportu nosaka modeļa struktūras atbilstība rezultātam - vai ir iespējams rezultātu iegūt, regulāri apstaigājot modeļa daļas. Ja regularitāte ir liela, tad atliek ar parastajiem līdzekļiem risināt mērķa formas iegūšanas jautājumus.

Tāpat kā pie importa arī šeit ir svarīgs modeļa attēlojuma jautājums (šajā gadījumā navigācija) - var darboties meta meta modeļa jēdzienos (universālā glabātuve) un modeļa jēdzienos (D-modelis). Universālās glabātuves navigācija aprakstīta [1]. D-modeļa navigācija ir atvasināta no D-modeļa struktūras (skat. [2]) :

- `public List<T> MDSgetD<T>(int container) where T:Dobject` ir viena veida objektu atlasīšanas metode ar neobligātu saturošā objekta nosacījumu (*container*)
- `public List<LogicalJoin.Jpart> MDSfind_Jpart(int container=0)` veida metodes (piemērs no *LogicalJoin* meta modeļa atrod visas *Jpart* tipa entītijas) ir konkrēta D-modeļa analogs iepriekšējai metodei. Šāda veida metodes iznāk gan efektīvākas gan arī nodrošina kontroli jau kompilācijas laikā.
- `public List<LogicalJoin.Jpart> @JpartS` ir *LogicalJoin* meta modeļa atbilstošajā D-modelī klases *Join* īpašība, kas dod atbilstošās sastāvdaļas. Arī šī patiesībā ir iepriekšējo metožu variācija (jāatzīmē, ka ļoti ērta), lai atrastu kāda tipa apakšobjektus (*contains*). Šādas īpašības ar vārdu - apakšobjekta tipa vārds, kuram galā pielikts "S", - tiek noģenerētas visām *contains* relācijām meta modelī.
- `public string @predefinedKind` veida īpašības tiek noģenerētas visiem skalāriem meta atribūtiem. Šādā veidā tiek iegūts ļoti ērts atribūtu vērtību nolasīšanas (daudzvērtīgiem atribūtiem dod atpakaļ pirmo vērtību) veids (starp citu, *setter* pievieno jaunu vērtību).
- `public LogicalJoin.Jpart @main` veida īpašības tiek noģenerētas visiem saites (*link*) meta atribūtiem. Šādā veidā tiek iegūts ļoti ērts piesaistītā objekta nolasīšanas (daudzvērtīgiem atribūtiem dod atpakaļ pirmo vērtību) veids (*setter* nav pieejams).

Tālāk nedaudz detalizētāk aplūkosim tipiskākos eksportu veidus.

3.1. No universālās glabātuves uz modeļu apmaiņas formātu.

Tā kā šajā gadījumā visa nepieciešamā strukturālā informācija ir pieejama, tad (tāpat kā importa gadījumā) projekta ietvaros tika izveidots eksporteris. Tehniski darbs tika sadalīts 2 daļās:

1. Koka, kas atbilst izvadāmajam XML (tā sauktais X-modelis) izveide. Tā kā datu glabātuve dod tiešu pieeju pēc identifikatora jebkuram modeļa un arī meta modeļa objektam, tad varēja izveidot rekursīvu algoritmu, kas atkarībā no objekta tipa veidoja atbilstošos X-modeļa apakškokus. Ārpus pamata algoritma tika izstrādāts kods, lai atlasītu objektus, uz kuriem tiešām ir atsauces, un šo informāciju saglabātu darba struktūrās.
2. No X-modeļa atbilstošā XML ģenerācija. Samērā tieša pārkodēšana ar vienīgo interesanto momentu, ka atbilstoši 1. solī izveidotajai papildus datu struktūrai objektu tehniskie identifikatori (*id*) tiek izvadīti tikai tiešām referencētiem objektiem. Šādā veidā tipiskie modeļu faili iznāk jūtami mazāki un ir ērtāk lasāmi (palīdz arī tas, ka ir vienkārša pazīme, vai uz objektu ir kāda atsauce).

Starp citu, nav nepieciešams speciāls eksports no D-modeļa uz modeļu apmaiņas formātu, jo D-modelis patiesībā ir tieša virsbūve uz universālās glabātuves un var tikt izmantots šis pats eksporteris.

3.2. Transformāciju valodā veidots eksports.

M2M transformācijas ([3]) ir pamatā orientētas uz divu (izejas un mērķa) modeļu situāciju, bet pieļauj arī patiesībā viena modeļa kontekstu. Transformāciju valoda dod samērā labus līdzekļus modeļa apstaigāšanai (nav pat jāprogrammē cikli), bet ģenerējošā daļa ir orientēta uz jaunu objektu veidošanu. Izmantojot *lightC* konstrukciju, iespējams noprogrammēt atbilstošo izvadīšanas kodu.

Šī ir vieta, kur transformāciju valodu varētu dabiski paplašināt ar teksta ģenerācijas konstrukcijām, par paraugu ņemot T4 templates ([4]) vai atskaišu ģeneratorus.

3.3. T4 ģenerācija.

Ja iepriekšējais eksporteru paveids uzsvaru lika uz palīdzību modeļa apstaigāšanā, bet bija neērts ģenerācijas daļā, tad šinī veidā ir otrādi. Microsoft T4 tehnoloģija ([4]) koncentrējas uz C# konstrukciju uzskatāmu savienošanu ar ģenerējamo tekstu. Šī pieeja izmantojama gan ar universālo glabātuvi gan ar D-modeli - programmētājam jālieto atbilstošās modeļa apstaigāšanas metodes.

Pielietojumu projektu koda ģenerācijai šī liekas visatbilstošākā pieeja, jo ir viegli uztverama (ielādējam modeli un staigājot pa to T4 tipiskā veidā ražojam mērķa kodu) un tehnoloģiski pilnīgi atbilst Visual Studio loģikai darbā ar templatēm.

Patiesībā šāda pieeja tiek plaši izmantota arī MEDUS projekta sastāvdaļu izstrādē, tādejādi MEDUS tehnoloģiju izmantojot pašas tehnoloģijas izstrādē (pazīstamais *bootstrapping* princips).

3.4. Specifisks eksports.

Gadījumos, kad kaut kādu apstākļu dēļ iepriekšminētās pieejas neder, var izveidot eksportu pilnīgi manuāli. Respektīvi izmantojot datu glabātuves navigācijas metodes (tāpat kā T4 gadījumā) un manuālu izvadīšanas kodu (tāpat kā transformāciju valodas gadījumā), tiek izveidots pilnīgs C# kods ar nepieciešamo funkcionalitāti.

4. Rezultāti

Aktivitātes ietveros tika izstrādāts modeļu eksports – no universālās glabātuves uz moeļu apmaiņas formātu, transformāciju valodā veidots eksports un T4 ģenereācija.

5. Literatūras saraksts

[1] Nr. 1.3.1 "Navigācijas nodrošināšana"
[2] Nr. 2.3.2 "Modeļa importa izveide"
[3] Nr. 1.3.2 "Modeļa transformāciju atbalsta izstrāde (M2M transformāciju atbalsts)"
[4] http://en.wikipedia.org/wiki/Text_Template_Transformation_Toolkit

6. Pielikumi.

6.1. T4 bāzēta eksporta piemērs.

```
<#@ include file="./../../../_/TT/DataViews_dll.ttinclude" once="true" #>
<#@ output extension=".cs" #>
<#@ Assembly Name="System.Core" #>
<#@ Assembly Name="System.Windows.Forms" #>
<#@ Assembly Name="$(ProjectDir)/../../../_/bin/MEDUS.dll" #>
<#@ Assembly Name="$(ProjectDir)/../../../_/bin/DataViews.dll" #>
<#@ import namespace="System" #>
<#@ import namespace="System.IO" #>
<#@ import namespace="System.Diagnostics" #>
<#@ import namespace="System.Linq" #>
<#@ import namespace="System.Collections" #>
<#@ import namespace="System.Collections.Generic" #>
<#@ import namespace="MEDUS" #>
<#@ import namespace="DataViews" #>
<#



MDSmo_DataViews model =
new MDSmo_DataViews(loadXmodel("DataViews.xml", getLUpath()),
                    MEDUS.Model.ExplorePolicy.ignore);


#>
using System;
using System.Linq;
using Microsoft.VisualStudio.TestTools.UnitTesting;
using MDSDB_DUS;
using MEDUSP;
using MEDUSR;
using MEDUSBL;

namespace Protons.Tests
{
   [TestClass]
   public class Selects
   {
<# foreach (DataView v in model.MDSfind_DataView()){ #>
       [TestMethod]
       public void select<#= v.name #>()
```

```
        {
                    IAuthManager am = new LocalAuthManager();
            am.Login("user", "user100", false);
                        var cc =  (new  <#= v.name #>()).startSelect().Take(10).ToList();
            MEDUSContext tc = new MEDUSContext();
                    var cc1 =  (new  <#= v.name #>()).startSelect(tc).Take(10).ToList();
                        am.Logout();
        }
<# } #>

    }
}
```