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.3.2 "Biznesa lēmumu atdalīšana no izstrādes lēmumiem" progresa pārskats

Pārskats Nr. 13 par periodu no 2014.gada 1.janvāra līdz 2014.gada 30.jūnijam.

SATURS

1. Kopsavilkums....3
2. Ievads....4
3. Biznesa modelis....5
   3.1. Veidošana....5
   3.2. Izmantošana....6
   3.3. Paredzamās problēmas....7
4. Izstrādes pieejas....8
   4.1. CASE rīki....8
   4.2. Multi-modeļu pieeja....8
5. Literatūras saraksts....12

1. Kopsavilkums

Pārskata periodā (2014-01-01 – 2014-06-30.) projekta „Multi - modeļu izstrādes tehnoloģija .NET pielietojumu projektiem” aktivitātes Nr.3.2 "Biznesa lēmumu atdalīšana no izstrādes lēmumiem" ietvaros veikti šādi darbi:

1. Programmatūras standartu izpēte.
2. Biznesa modeļa izpēte, tai skaitā, biznesa modeļu veidošana, izmantošana, mainība.
3. CASE rīku izpēte (GRADE, ErWIN, ARIS, MetaEdit+, Eclipse, RSA).
4. DFD (data flow diagramm), SADT (Structured Analysis and Design techniq), ERD (Entity Relationship Diagramm) metadoloģiju izpēte.
5. UML modelēšanas valodas izpēte.
6. Multi-modeļa pieejas izstrāde, kas ietver loģisko modeļu identifikāciju, tehnisko modeļu identifikāciju, kā arī paredzamo problēmu identifikāciju.

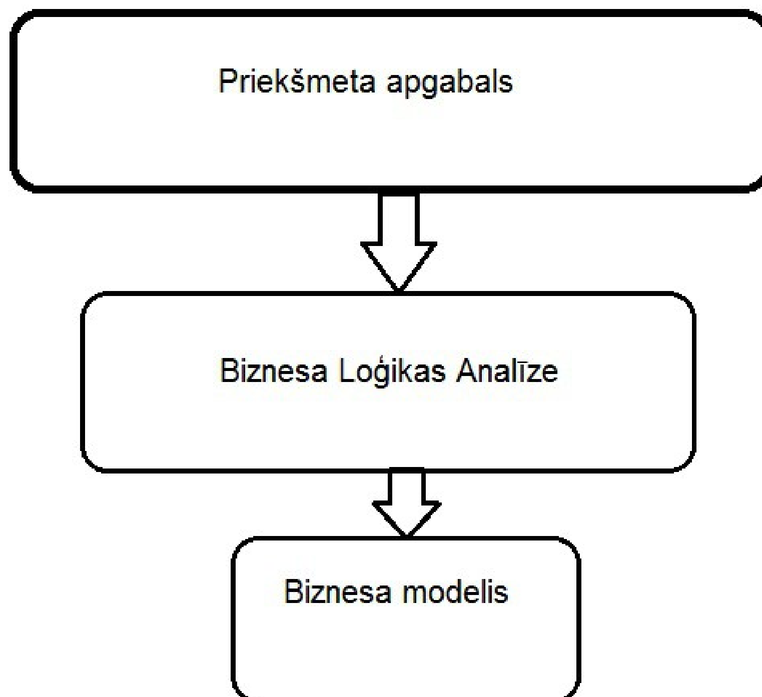
2. Ievads

Modernās izstrādes tehnoloģijas un to atbalstošie ietvari dod plašu iespēju lietojumprogrammas veidošanā gandrīz jebkurā priekšmetu apgabalā. Tieši priekšmetu apgabals nosaka biznesa lēmumus un prasības programmatūrai, kā arī tehniskā risinājuma izvēli. Tajā pašā laikā, viena no galvenajām lietojumprogrammu izstrādes problēmām ir augstā mainība visās svarīgākajās projekta izstrādes sastāvdaļās, tas ir, priekšmetu apgabalā un tehniskajā izpildījumā.

3. Biznesa modelis

3.1. Veidošana

Atbilstoši programmatūras izstrādes standartiem (J-STD-016, IEEE 12207, LVS 68:1996 u.c.)[1], jebkura projekta sākumā, jāveic biznesa analīzi, lai fiksētu un formalizētu prasības topošai lietojumprogrammatūrai.



Zīm.1

Biznesa lēmumu/prasību noformēšana ar modelēšanas valodas līdzekļiem veido modeli. Modelis kalpo:

- lai vizualizētu un ar to atvieglotu priekšmeta apgabala uztveri, analīzi;
- lai formalizētu biznesa lēmumus, padarītu prasības viennozīmīgās un nepretrunīgās;
- lai ierobežotu priekšmeta apgabalu, noskaidrotu, kas tiks implementēts topošajā lietojumprogrammatūrā, un kas paliks ārpus sistēmas;
- tehnoloģisko risinājumu izstrādei.

Jebkura projekta biznesa prasības var sadalīt divās grupās:

- prasības, kuras ir līdzīgas (kopīgas) vairākiem projektiem (biznesa objekta ievade/izvade, lietotāju autorizācija/autentifikācija, audits, vēsturiskas informācijas pārvalde utt.)
- prasības, kas ir raksturīgas konkrētajam projektam (biznesa objekti, raksturīgie tieši šim apgabalam, specifiskas funkcionālās prasības, piemēram noteikto rādītāju aprēķināšana).

Visas šīs prasības tiek atspoguļotas biznesa modelī.

Programmatūras izstrādē pašlaik ir vairākas pieejas modeļa veidošanā. Klasiskajā ūdenskrituma modelī [2] tiek definētas projekta prasības, kas ir stingri fiksētas sākotnējās projekta izstrādes stadijās. Līdz ar to modeļa mainīšana un izmantošana ir augstākā mērā sarežģīta.

Mūsdienās, prasības, kas ir raksturīgas priekšmetu apgabalam arī nepārtraukti mainās. Tas noved pie nepieciešamības ātri un saskanīgi mainīt biznesa modeli, uz kura balstās projekts, un attiecīgi programmatūras kodu. Modernā „Agile" metodoloģija [3] ir mēģinājums tikt galā ar nepārtraukti mainīgajām prasībām. Ūdenskrituma modelis paliek stratēģiskajai projekta plānošanai, bet konkrētie lokālie uzdevumi tiek noteikti kā „sprint"(iterācija) „scrum" metodoloģijā , lai projekts virzītos uz savu mērķi.

Visās šīs izmaiņas bieži vien ir ļoti sarežģīts, darba un materiālo resursu ietilpīgs pasākums.

3.2. Izmantošana

Biznesa modelis tomēr nav pašmērķis, un domāts lai kalpotu par pamatu koda veidošanai projektā. Tam ir vairāki aspekti:

- mainās biznesa modelis – mainās programmatūras kods;
- ja mainījās kopīgās prasības vairākiem projektiem, nācās veikt izmaiņas vairākās vietās(projektos);
- arī tehnoloģiju jomā situācija ir problemātiska – izmantojamie standartrisinājumi ļoti strauji mainās. Tā rezultātā katrs nākošais projekts ir jāizpilda citā tehnoloģijā.;
- īpaši nepatīkams aspekts ir daudzkāršās mantošanas izzušana no moderno valodu spektra (piemēram C#). Ņemot vērā, ka dažāda veida tehnoloģijas, kas tiek izmantotas projektu ietvaros, bieži „aizņem" mantošanu saviem tehniskajiem

mērķiem, tad projekta izstrādātāji zaudē ļoti būtisku objektorientācijas īpašību – mantošanu.

3.3. Paredzamās problēmas

Strauji mainīgās projekta izstrādes tehnoloģijas ļoti iespaido biznesa prasību realizāciju. Ilgstoša .NET tehnoloģijas izmantošana rāda, ka pārejot no versijas uz citu versiju, mainās gan tā koncepcija, gan tehniskais izpildījums. Pazūd un parādās citi efektīvie tehnoloģiskie risinājumi. No otras puses, biznesa prasību mainīšana arī iespaido lietojumprogrammatūras kodu. Tas viss noved pie atkārtotas projekta koda pārrakstīšanas, kas ir darbietilpīgs process. Mūsu pieeja, kas tālāk tiks izklāstīta, ir vēl viens solis šīs problēmas risināšanas virzienā.

4. Izstrādes pieejas

4.1. CASE rīki

Viens no virzieniem augšminēto problēmu risināšanas jomā ir biznesa CASE rīki [4] ar koda ģenerāciju. (piemēram GRADE, ErWIN, ARIS, MetaEdit+, Eclipse, RSA). Galvenais CASE-tehnoloģiju mērķis ir programmatūras projektēšanas procesa norobežošana no kodēšanas procesa un turpmākajām attīstības stadijām, maksimāli automatizējot projektēšanas procesu. Lai sasniegtu mērķi CASE-tehnoloģija izmanto divas, principiāli atšķirīgas pieejas projektēšanai: strukturētā un objektu orientētā.

Strukturētā pieeja paredz uzdevuma dekompozīciju(sadalīšanās) uz funkcijām, kuras jāautomatizē. Savukārt, funkcijas ir arī sadalītas apakšfunkcijās, uzdevumos un procedūrās. Rezultātā veidojās sakārtotā funkciju hierarhija ar pārsūtamo starp funkcijām informāciju. Galvenās izmantojamas metodoloģijas šajā pieejā ir: DFD (data flow diagramm), SADT (Structured Analysis and Design techniq)[5], ERD (Entity Relationship Diagramm).

Objektu orientētā pieeja balstās uz UML valodas izmantošanas. UML(*Unified Modeling Language*)[6] ir modelēšanas valoda, kur katrs modelējamais apgabals tiek aprakstīts vairākos abstrakcijas slāņos, izmantojot vairākus diagrammu tipus. Unificēts modeļa apraksts dot plašu iespēju koda ģenerācijai.

Diemžēl šim pieejām arī ir savi trūkumi. No vienas puses, praktiski nav iespējams izstrādāt universālu koda ģeneratoru, kas derētu pietiekoši lielam projektu skaitam mūsdienu mainīgajos apstākļos. Savukārt, bez koda ģenerācijas zūd jēga modelim, jo tas vairs automātiski nesaistās ar programmatūru un rezultātā visas izmaiņas jāveic divas reizes – modelī un programmatūrā. Papildus ir grūti definēt modeļa struktūru, kas derētu visiem projektiem. Jo plašāku projektu loku noklāj modelis, jo sarežģītāks tas ir. Kā rezultātā, modelī ietilpst daudz parametrizējāmu tehnisku atribūtu, un modeļa aprakstīšana un izmantošana kodu ģenerācijā kļūst pārāk sarežģīta.

4.2. Multi-modeļu pieeja

Mūsu pieejas pamatideja ir viena sarežģīta modeļa vietā lietot vairākus modeļus. Modeļa struktūru nosaka tā meta modelis, kas ir stingri orientēts uz veicamo uzdevumu. Lai padarītu katru modeli vairāk maināmu un neatkarīgu, tiek piedāvāts sadalīt modeļus 2 grupās (sk. Zīm.2):

- Loģiskie modeļi (LM) – aprakstošie biznesa modeļi, kurus izveido biznesa analītiķi vai dizaineri uc., lai formalizētu sistēmas objektus, standartstruktūras, to parametrus utt. Šie modeļi ir atkarīgi no biznesa apgabala, un būtu vislabāk to

aprakstīt biznesa loģikas terminos. Vai loģiskais modelis ir tikai viens vai vairāki, varētu būt atkarīgs no projekta izstrādātāju vēlmēm.

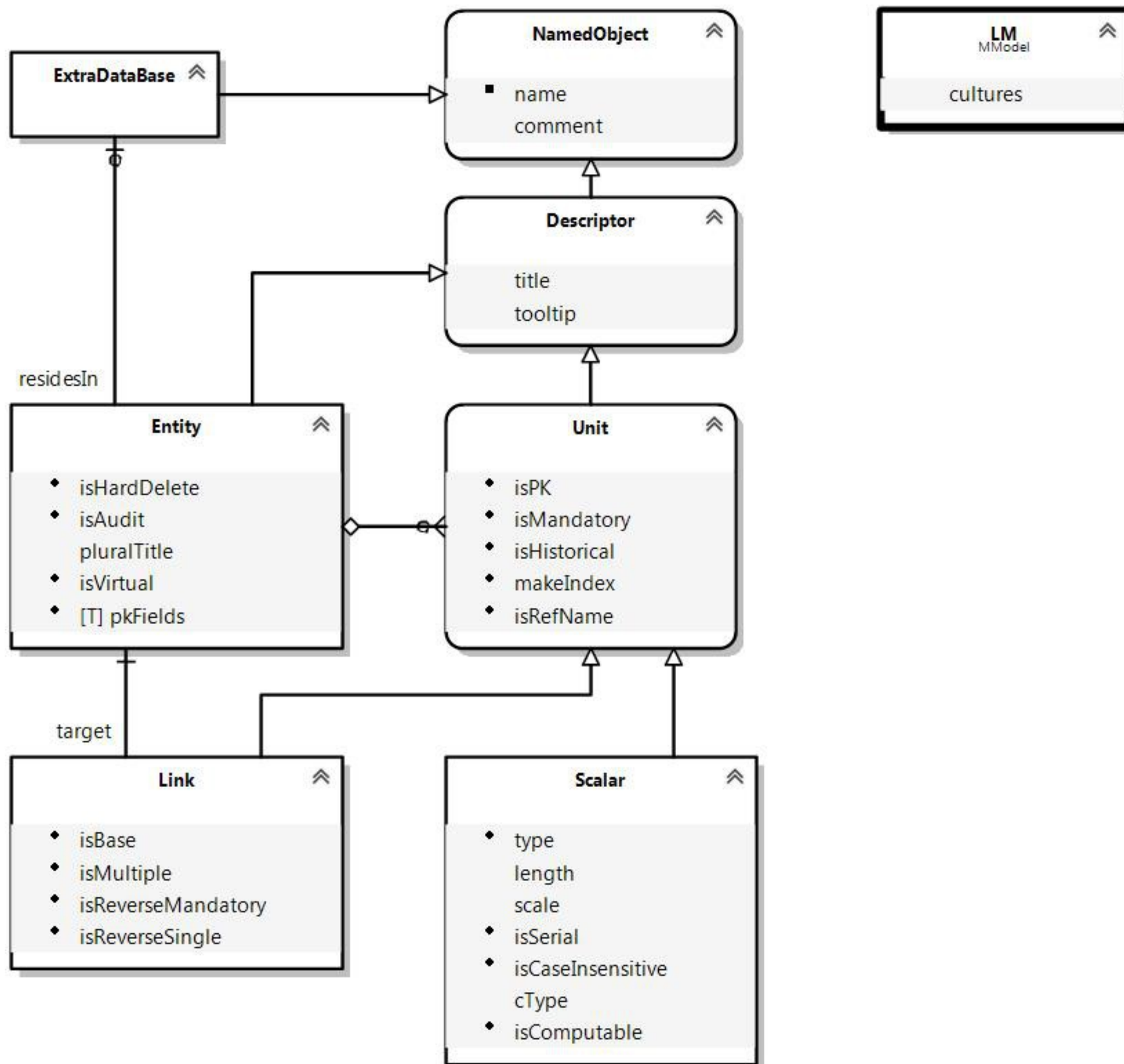
- Tehnoloģiskie modeļi (TM) - modeļi tiek izmantoti gatavā programmprodukta ražošanā - koda ģenerācijā /interpretācijā. Būtiskākais šiem modeļiem ir tas, ka viņi ir konkrētā tehnoloģiskā risinājuma apgabala specifiski (piemēram, tabulu modelis, join- modelis, darba vietu modelis).

LM un TM ir dažādi metamodeļi, dažādu priekšmeta apgabalu dēļ. Pietiekoši formalizēts LM dot iespēju, koda maināmo daļu producēt automātiski. Maksimāli atdalot biznesa loģiku loģiskajā modelī, var panākt, ka izmaiņas biznesa loģikā prasīs minimālas pūles implementācijā. Modeļu transformācijas nodrošina informācijas pārvietošanu starp daudzajiem modeļiem.

Zīm.2

Apskatīsim loģiska modeļa piemēru. (sk. Zīm. 3). Zīmējumā zemāk attēlota vienkāršota loģiska modeļa struktūra (lietotie apzīmējumi izskaidroti [7]).

Šī loģiskais modelis apraksta iespējamo biznesa lietojumprogrammas moduli: modeļu redaktora daļu, kas izveido un rediģē lietotāja modeļus. Galvenais modeļa objekts ir *Entity*, kas satur objektus *Link* un *Scalar*. Objekts *ExtraDataBase* ļauj *Entity* objektu piekārtot konkrētai datubāzei.



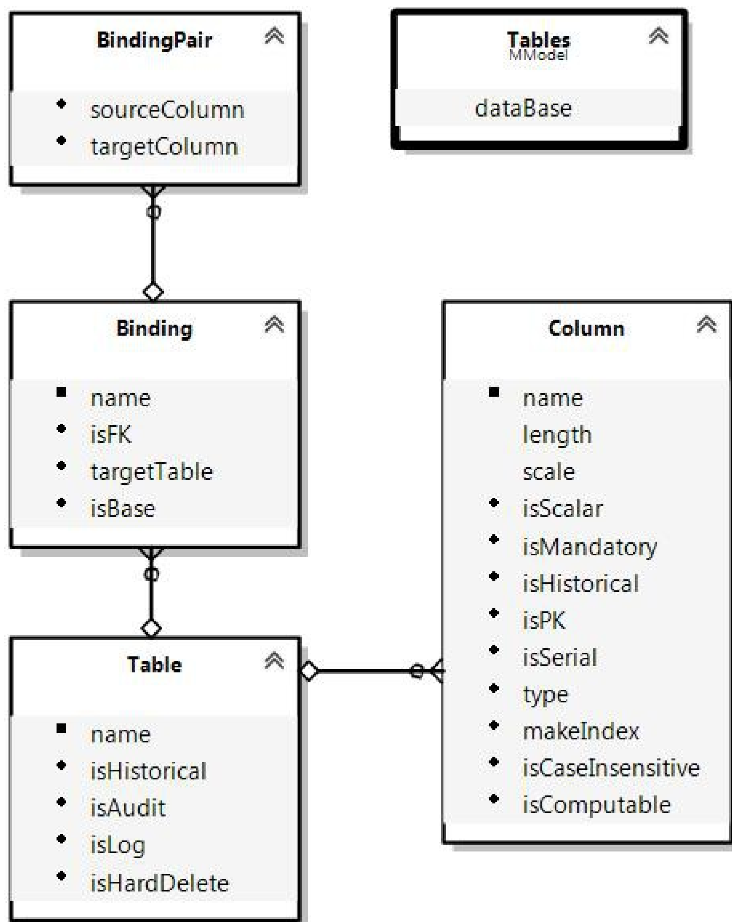
Zīm.3

Savukārt, šim loģiskā modeļa piemēram atbilst tehniskā modeļa piemērs (sk. Zīm.4), kuru iegūst ar tehniskas transformācijas palīdzību (lietotie apzīmējumi izskaidroti [7]).
Šajā tehniskajā modelis atbilst izvēlētam tehnoloģiskam risinājumam – glabāšana datubāze:

- *Table* objekts atbilst *Entity* objektam ar visiem mantotiem atribūtiem;
- *Column* objekts atbilst *Link* vai *Scalar* objektiem ar viesim mantotiem atribūtiem;
- *Binding* un *BindingPair* objekti, balstoties uz informācijas no loģiska modeļa, apraksta saites starp *Table* objektiem (datubāzes tabulām).

Jāpiebilst, ka pilnīgi atdalīt loģisko no tehniskā nav iespējams. Ir paredzams, ka nāksies ieviest objektus vai atribūtus loģiskajā modelī tikai tāpēc, lai nodrošinātu tehniskais modelis ar nepieciešamo informāciju. Šo informāciju nepieciešamību nosaka izvēlētais tehnoloģiskais risinājums. Viens no tāda veida informācija piemēriem, mūsu gadījumā, ir atribūts *dataBase* pie objekta *Tables*, kas apraksta modeli. Šī atribūta vērtība tiek iegūta no loģiska modeļa objekta *ExtraDataBase* atribūta *name*, kas ir datubāzes nosaukums. Ja

loģiskajā modeli ir aprakstīti *Entity* tipa objekti, kuri pieder, piemēram, divām dažādām datubāzēm, tad tehniskas transformācijas laikā, tiks uzbūvēti divi tehniskie modeļi – katrs savai datubāzei, aprakstot tieši šīs datubāzes tabulas.



Zīm. 4

Kā no tehniskā modeļa iegūt lietojumprogrammas kodu ir koda ģenerācijas jautājums un ir stingri saistīts ar projektā izvēlēto tehnisko risinājumu un izmantojamo programmatūras veidošanas tehnoloģiju.

5. Literatūras saraksts

[1] https://www.lvs.lv/ - Latvijas Standarts;
http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?reload=true&punumber=4475822 - IEEE ***Xplore*** Digital Library;

[2] http://www.waterfall-model.com/ - Waterfall Model (Ūdenskrituma modelis)

[3] http://www.agilemodeling.com/ - Agile Modeling („Agile" modelēšana)

[4] http://en.wikipedia.org/wiki/Computer-aided_software_engineering - CASE Tools (Case rīki)

[5] William S. Davis (1992). *Tools and Techniques for Structured Systems Analysis and Design*. Addison-Wesley. ISBN 0-201-10274-9

[6] http://www.omg.org/spec/UML - Unified Modeling Language™ (UML®)"

[7] Projekta aktivitātes Nr.1.1 "Meta metamodeļu izpēte" progresa pārskats.