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.2.3.2 "Modeļa importa izveide" progresa pārskats

Pārskats Nr.17. 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.2.3.2 "Modeļa importa izveide" ietvaros veikti šādi darbi:

1. Modeļu importa prasību analīze.
2. Importa no universālā modeļu glabāšanas formāta uz universālo glabātuvi izstrāde.
3. Importa no universālā modeļu glabāšanas formāta uz D-modeli izstrāde.
4. Pielietojumam specifiska importa izpēte.
5. Aktivitātes pētnieciskā darbība apspriesta ik nedēļas projekta semināros.

2. Ievads

Šis pārskats ir veltīts projekta apakšaktivitātes Nr.2.3.2 " Modeļa importa izveide" ietvaros paveiktajam. Apakšaktivitātes ietvaros tika izstrādāts modeļu importētājs.

Pārskatā aprakstītas datu glabātuves un importa paveidi – no universālā modeļu glabāšanas formāta uz universālo glabātuvi, no universālā modeļu glabāšanas formāta uz D-modeli un pielietojuma specifisks imports.

3. Importa izveide.

Imports jeb T2M transformācija nozīmē ieejas datu analīzi atbilstoši izvēlētajam ievaddatu formātam un tālāku to ievietošanu modeļa glabātuvē. Glabātuves papildināšanas atbalsts būtiski var ietekmēt importa programmas sarežģītību, tāpēc šis jautājums ir ļoti svarīgs un arī aplūkojams šī izstrādes etapa kontekstā.

3.1. Datu glabātuve.

Modeļa datu glabātuve var būt gan universāla (der jebkuram meta modelim), gan specifiska vienam konkrētam meta modelim. Universālās krātuves priekšrocība ir tās universālums, savukārt meta modeļa specifiska krātuve - tiks saukta par D-modeli - ļauj darbības rakstīt specifiskā meta modeļa jēdzienos.

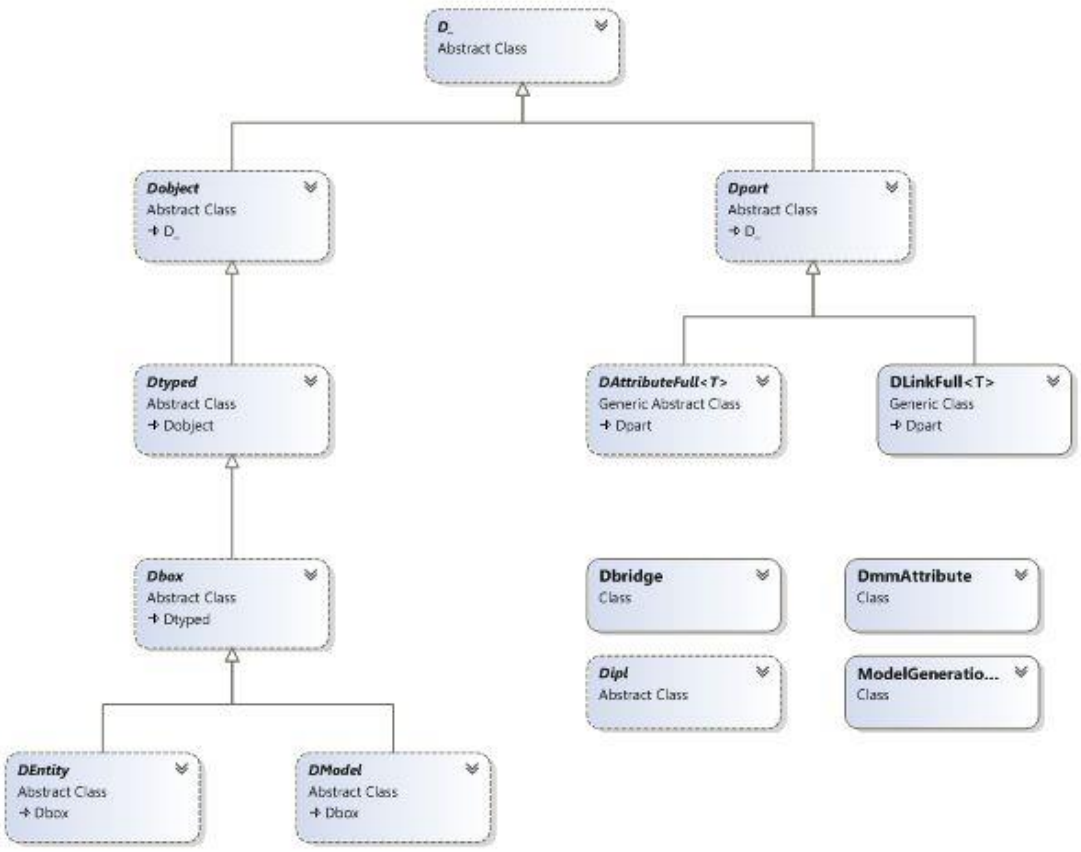
3.1.1. Universālā glabātuve.

Universālā krātuve patiesība jau diezgan detalizēti aprakstīta dokumentā [1]. Savukārt navigācija pa to (lai atrastu vietu, kur pievienot datus) aprakstīta [2]. Visvairāk importam svarīgā elementu pievienošana aprakstīta [3]. Tātad importam nav nepieciešami papildus primitīvi universālās krātuves gadījumā.

3.1.2. D-modeļi.

D-modeļi izstrādāti, lai dotu C# programmētājam iespēju darboties meta modeļa jēdzienos (darbošanās universālajā meta meta modeļa līmenī ir interpretatīva un līdz ar to grūti uztverama). Pamatideja ir ļoti vienkārša - atbilstoši meta modeļa definīcijai (skat. [5]) tiek izveidots C# kods, kas ļauj veikt nepieciešamās manipulācijas meta modeļa jēdzienos, un ir pārejas (wrapper) kods uz universālo glabātuvi. Tātad jebkuram D-modelim apakšā ir universālā glabātuve un tās interfeiss ir pieejams paralēli D-modeļa interfeisam.

Tehniski pāreja sastāv no standarta atbalsta koda (skatīt zīmējumu 1) un no modeļa definīcijas ģenerēta koda (savā ziņā vēl viena M2T transformācija). Saturīgi standarta daļā ir "D" klases atbilstoši "E" klasēm universālajā glabātuvē (Entity, Model, Attribute un Link), zem kurām kā apakšklases tiks pieģenerētas no konkrētā meta modeļa objektiem atbilstoši meta meta tipiem.



Zīmējums 1. D-modeļu pārejas standarta daļa.

Tīri tehnoloģiski D-modelis tiek veidots kā dinamiskā bibliotēka (DLL) ar fiksētu augšējo interfeisu *Dipl*, kas dod iespēju to viegli izmantot dažādos kontekstos (uz šī balstās D-modeļu izmantošana modeļu redaktorā [6]). Jāņem vērā, ka viena no projekta centrālajām koncepcijām ir meta modeļu mantošanas atbalsts - atbilstoši veidojas arī bibliotēku savstarpējās atkarības. Ņemot vērā, ka meta entītija var tikt papildināta apakšmodelī, jāatrisina problēma, ka atbilstošajai meta entītijas implementācijas klasei jāuzvedas kā meta klases implementācijai, ja šis D-modelis ir pēdējais hierarhijā, un kā meta klases implementācijai pārējos gadījumos. Tā kā Microsoft .NET realizācija neļauj brīvi papildināt dažādos moduļos (assembly) esošas klases, tad nākas organizēt viltīgu funkcionēšanas shēmu. Pats svarīgākais pielietojumiem ir tas, ka vienmēr jālieto viszemākais D-modelis (no C# viedokļa varētu izsaukt metodes arī no augstāka līmeņa modeļiem, kas novestu pie nepareizas funkcionēšanas).

D-modelim ir vairākas metodes, kas svarīgas tā piepildīšanai:

- `public virtual DEntity MDScreateE(string type, int container)` tiek izmantota, lai pievienotu jaunu *DEntity* (kas saturīgi ir *EEntity*) modelim. Parametrs *type* ir

meta entītijas vārds, kuram atbildīs jaunais modeļa objekts un *container* ir saturošā objekta identifikators.

- `public virtual T MDScreateE<T>(int container=0) where T: DEntity` ir iepriekšējās metodes virsbūve ar tipa kontroli jau kompilācijas laikā.
- `public virtual EAttribute MDScreateA(string type, string value)` ir *DBox* metode atribūta vērtības (*EAttribute*) pievienošanai. Parametrs *type* ir meta atribūta vārds, kuram atbildīs jaunais atribūts. Šai metodei alternatīvs variants ir izmantot DmmAttribute klases apakšklasi (tiek noģenerēta) un tajā esošo vērtības piešķiršanu (setter). Šī alternatīvā metode kodā ir daudz uzskatāmāka un ļauj veikt kontroli jau kompilācijas laikā, bet neļauj izveidot daudzvērtīgu attribūtu.
- `public virtual ELink MDScreateR(string type, int target)` ir analoģiska metode saites vērtības pievienošanai. Arī šeit ir alternatīvā metode caur klases DLinkFull apakšklasēm ar tādām pašām priekšrocībām un trūkumiem.

D-modeliskās iepildīšanas metodes neizslēdz tiešo universālo metožu izmantošanu, kas būtu jādara ļoti uzmanīgi. D-modeļa attēlojumā atmiņā ir arī atbilstoša D-informācija, kas rodas tikai izmantojot paplašinātās metodes. Ja pievienošanai izmanto bāzes metodes, tad dažos gadījumos šī papildus informācija netiek veidota, kas savukārt var radīt problēmas D-modeliskā navigācijā.

3.2. Importa paveidi.

Kā jau sākotnēji tika atzīmēts, importa realizācija ir atkarīga gan no ievada formāta gan no datu krātuves veida. Tehnoloģiski ļoti svarīgs ir universālais modeļu kodējums, kas aprakstīts [4], bet dažādu pielietojumu veidiem ērtāki varētu būt citi formāti, kas parasti aplūko informāciju meta modeļa jēdzienos.

3.2.1. No universālā modeļu glabāšanas formāta uz universālo glabātuvi.

Tā kā universālā formāta importā visa nepieciešamā informācija (gan analizējamais formāts gan datu krātuves interfeiss) ir zināma, tad šīs aktivitātes ietvaros tika realizēts konkrēts importa algoritms.

Microsoft rīki dod iespēju atbilstoši XSD automātiski izveidot C# kodu, kas XML failu izanalizē un izvieto atbilstoši XSD veidotās struktūrās atmiņā. Šī metode tika izmantota pirmā importa soļa izveidei. Bāzes klase visiem XML atbilstošajiem elementiem ir *Obis*, un apakšklases veidojas pilnīgi atbilstoši XSD (skat. [4]):

- Attribute
- Box
- Comment
- Message
- Model
- National
- Reference

Pirmajā tuvinājumā XSD atbilstošie objekti ir līdzīgi ar universālās glabātuves modeļa daļas elementiem (kas arī ir dabiski), bet tas nenozīmē, ka informācijas pārvietošana ir triviāla.

Galvenās grūtības veido pāreja no XML kokveida struktūras uz relāciju datu bāzei līdzīgo glabātuves struktūru - XML kokā daļa saišu (*contains*) attēlojas kā iekļaušana, bet pārējās tiek kodētas kā objektu identifikatori un atsauces uz tiem. Tādēļ apstrāde jāveic 2 etapos:

- Pamata objektu veidošana. Tiek izveidoti objekti atbilstoši XML struktūrai (no saitēm aizpildās tikai *contains*). Šajā etapā tiek uzbūvētas tehniskas datu struktūras nākošā etapa veikšanai - objektu tabula ar identifikatoriem un neatrisināto saišu tabula.
- Saišu atrisināšana. Šajā etapā tiek izveidotas neatrisinātās saites. Normālā gadījuma XML *Reference* objektā izmantotais mērķa objekta identifikators atrodams iepriekšējā solī izveidotajā objektu tabulā un ir tehniski aizvietojams ar atsauci uz datu krātuves objektu. Jāņem vērā, ka modeļa XML failā ir iespējamas kļūdas (identifikators nav atrasts objektu tabulā vai rāda uz neatļauta tipa objektu), kuru gadījumā jādod diagnostika.

Ielādējamo datu atbilstība meta modelim veido importēšanas programmas specifisku aspektu. Jebkurš imports tiek veikts atbilstoši kādam meta modelim (parasti tiek izmantots meta modelis, kura vārdu satur modeļa XML fails). Optimistiskā gadījumā ielādējamo datu meta informācija - parasti, atribūts *type* - pilnīgi atbilst izmantojamajam meta modelim. Realitātē var izrādīties, ka atbilstība nav:

- Ielādējamie dati ir neatrisināmā pretrunā ar meta modeli (piemēram, datus ir skalārs atribūts, bet meta modelī ar šādu vārdu paredzēta saite). Šādos gadījumos ir realizēta avārijas (*Exception*) signalizācija. Neatbilstošo datu ignorēšanas iespēja tika atmesta kā pārāk bīstama (lietotājam tiek noslēptas būtiskas saturīgas problēmas).
- Ielādējamie dati nav paredzēti meta modelī (piemēram, XML failā ir atribūts, kas meta modelī nav paredzēts). Šajā gadījumā iespējami vairāki scenāriji (kas arī tika realizēti):
    - Avārija. Šis ir visvienkāršākais risinājums, kas parasti tiek lietots stingri fiksētu datu struktūru gadījumos. Tā kā izveidojamā tehnoloģija pretendē uz lielu adaptācijas pakāpi, tad tas varētu nebūt vēlamais variants.
    - Ignorance. Nepazīstamie dati tiek ignorēti. Šī metode parasti liekas vispievilcīgākā, jo ļauj darboties ar datiem, kuri meta modelī aprakstīti tikai daļēji (vēl nav pabeigta meta modeļa izveide vai tieši otrādi - modeļa failā palikuši dati, kas atbilst novecojušai meta modeļa versijai). Vienīgais trūkums šai pieejai ir tas, ka nepazīstamie dati nenonāk krātuvē un līdz ar to netiks apstrādāti (galvenais, ka netiks eksportēti). Neraugoties uz šo trūkumu, ignorēšanas variants ir tipiski vispielietojamākais.
    - Apmācība. Šajā režīmā neatrastais meta modeļa elements tiek automātiski pievienots. Tas dod interesantas iespējas meta modeļu sintezēšanai no piemēriem, kas diemžēl iziet ārpus šī projekta rāmjiem.

3.2.2. No universālā modeļu glabāšanas formāta uz D-modeli.

Šis importa paveids loģiski ir tuvs iepriekšējam, bet jāveic, izmantojot D-modeļa papildināšanas metodes. Šīs metodes ir definētas kā virtuālas kopējās virsklasēs, kas dod iespēju veikt algoritma kodēšanu neatkarīgi no konkrētā D-modeļa teksta (tas tiks iedarbināts automātiski mantošanas ceļā). Saturīgi algoritms atkārto iepriekšējā nodaļā aprakstīto un visur, kur iespējams, izmanto tā C# kodu.

Tā kā šī importa rezultātā rodas D-modelis, kas ir daudz ērtāks lietošanā, tad šis ielādēšanas veids ir nospiedoši populārākais.

3.2.3. Pielietojuma specifisks imports.

Iepriekšējās nodaļās aprakstītie importētāji balstās uz specifisko modeļu apmaiņas formātu ([4]), kas nav tipiska situācija datiem, kas nāk no vides ārpus MEDUS projekta. Šādos gadījumos jāizveido atbilstošs analizators (diemžēl vispārīgā gadījumā neko līdzēt nav iespējams) un jāizmanto atbilstošās (universāls/D) iepildīšanas metodes.

Skaists piemērs ir transformatoru ģenerators ([7]), kurā imports tiek veikts no specifiskas valodas ([8]). Valodas sintaktiskajai analīzei tiek izmantota klasiskā YACC un LEX tehnoloģija ([9]), kur YACC akcijās veic D-modeļa papildināšanas metožu izsaukumus (ne tieši, jo pāreja no valodas konstrukcijām uz modeli nav triviāla).

4. Rezultāti

Aktivitātes ietveros tika izstrādāts modeļu imports – no universālā modeļu glabāšanas formāta uz universālo glabātuvi, no universālā modeļu glabāšanas formāta uz D-modeli.

5. Literatūras saraksts

[1] Nr.1.4 "Modeļu kodēšana un ielādes metožu izpēte" progresa pārskats

[2] Nr.1.3.1 "Navigācijas nodrošināšana" progresa pārskats

[3] Nr.2.3.1 "Elementu pievienošanas izstrāde" progresa pārskats

[4] Nr.1.2 "Universālā modeļa formāta izpēte" progresa pārskats

[5] Nr.1.1 "Meta metamodeļu izpēte" progresa pārskats

[6] Nr.2.2 "Universālā modeļa redaktora izstrāde" progresa pārskats

[7] Nr.1.3.3 "Modeļa apstrādes transformāciju mašīnas izstrāde, kas ietver daudz-modeļu manipulācijas" progresa pārskats

[8] Nr.1.3.2 "Modeļa transformāciju atbalsta izstrāde (M2M transformāciju atbalsts)" progresa pārskats

[9] http://en.wikipedia.org/wiki/Yacc