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.1.4 "Modeļu kodēšana un ielādes metožu izpēte" progresa pārskats

Pārskats Nr. 6 par periodu no 2014.gada 1.janvāra līdz 2014.gada 30.jūnijam.

SATURS

1. Kopsavilkums

Pārskata periodā (2014-01-01 – 2014-06-01.) projekta „Multi - modeļu izstrādes tehnoloģija .NET pielietojumu projektiem” aktivitātes “Modeļu ietvara pētniecība” apakšaktivitātes "Modeļu kodēšana un ielādes metožu izpēte" ietvaros veikti šādi darbi:

1. Veikta iespējamo modeļu apmaiņas formātu izpēte. Apakšaktivitātes ietvaros tika pētīti kompaktie un lasāmie formāti, pētījumi ietver:
   a. XML formāta priekšrocību un trūkumu izpēti;
   b. XML atbalsta .NET standarta bibliotēkās izpēti;
   c. JSON formāta izpēti.
2. Identificētas prasības modeļu apstrādes formātam, un veikta modeļu apmaiņas formāta loģiskās struktūras analīze.
3. Modeļu apmaiņas formāta izstrāde, kurā nodefinēta formāta pamatstruktūra – Model, Box, Reference, Attribute, National, kā arī formāta paplašinājumi – Atribūts „comment”, Comment, Message.
4. Modeļu apmaiņas formāta kodēšanas izpēte un myContainer, EModel, myType, Eentity, Eattribute, Enatianl un Elink izstrāde.
5. Aktivitātes pētnieciskā darbība apspriesta ik nedēļas projekta semināros.

2. Ievads

Šis pārskats ir veltīts projekta apakšaktivitātes Nr.1.4 "Modeļu kodēšana un ielādes metožu izpēte" ietvaros veiktajiem pētījumiem.

Sākotnēji tika veikta apmaiņas formātu izpēte, īpašu uzmanību veltot lasāmajiem apmaiņas formātiem XML un JSON, kas tika atzīti par piemērotākajiem projekta vajadzībām. Lai atrastu visatbilstošāko modeļu apmaiņas formātu, tika identificētas un prioritizētas modeļu apstrādes formāta prasības. Gala rezultātā priekšroka tika dota XML formāta izmantošanai.

Pētījumu rezultātā izstrādāti loģiskās struktūras izstrādes principi un izstrādāts uz meta metamodeļa instancēšanas bāzētu tolerantu formātu, kas ir vēl plašāks nekā meta metamodeļa vajadzībām nepieciešams.

Tā kā izstrādāta modeļu apmaiņas formāta definīcija ir vispārīgāka par to kā tiek definēti modeļi (meta metamodeļa un metamodeļa definīcijas uzliek papildus ierobežojumus) un modeli šajā formātā var iekodēt dažādos veidos, tad tika izstrādāts konkrēts kodēšanas veids.

3. Modeļu apraksta formāta izstrādes principi

3.1. Pamatprasības

Modeļu apmaiņas formātam jābūt tādam, lai nodrošinātu sekojošās pamatprasības:

1. visi modeļi, kas atbilst metamodeļiem, kas, savukārt, atbilst meta metamodelim [1], ir iespējami;
2. algoritmiski viennozīmīgi iespējams no modeļa apraksta iegūt kodējumu apmaiņas formāta failā un pretēji.

Šāda veida prasības ir visiem apmaiņas formātiem (mainās tikai 1. punktā definētais derīguma apgabals) un ir realizējamas daudz dažādos veidos. Jebkurš apmaiņas formāts parasti ir kompromiss starp sekojošām īpašībām (to svarīgums ir ļoti atkarīgs no pielietojuma):

1. formāta kompaktums;
2. mašīnapstrādes efektivitāte;
3. lasāmība cilvēkam;
4. formāta ģeneriskums;
5. viegla modificējamība.

3.2. Tehniskā pieraksta veidi

Tehniski datu apraksta formāti grupējas divos pamatveidos:

- kompaktie formāti;
- lasāmie formāti.

Mūsdienu tipiskākas shēmas tika analizētas, lai atbilstoši lasāmības un apstrādāšanas efektivitātes prasībām (svarīgs tipveida modeļa apjoms) izvēlētos atbilstīgāko.

3.2.1. Kompaktie formāti

No mašīnapstrādes viedokļa vislabākie ir kompaktie formāti, kuros metainformācija (saturu aprakstošā informācija) neparādās vispār (apstrādājošās programmas "zin", kur kas atrodas un kā to visu interpretēt) vai ir ļoti minimāli (piemēram, tikai kolonu virsraksti). Šāda veida formāti vēsturiski radās jau datoru lietošanas pirmsākumos un bija ļoti dažādi. Kā mūsdienu populārākos kompaktos formātus aplūko CSV ("comma separated values") saimi [2] . Tie parasti variējas ar kolonu virsrakstu klātbūtni/prombūtni, dienesta atdalošajiem simboliem un speciālo simbolu kodēšanu. Parasti viens CSV fails apraksta vienu "tabulu". Tā kā modeļi ir ar sarežģītu struktūru, tad, acīmredzot, nāksies vai nu lietot vairākas "tabulas" viena modeļa aprakstam, vai arī apraksts būs sastāvošs no maziem tehniskiem elementiem ar sarežģītu interpretāciju.

3.2.2. Lasāmie formāti

Cilvēkam lasāmie formāti arī laika gaitā ir evolucionējuši un ir dažādu paveidu. Galvenā to īpatnība ir tā, ka tie satur metainformāciju, kas atvieglo cilvēkam lasāmību, bet, protams, padara kodējumu neefektīvāku - gan garāku, gan sarežģītāk apstrādājamu. Šo

formātu liela priekšrocība ir iespēja paaugstināt fleksibilitāti - apstrādājošas programmas no "cieti iešūtām" (iekompilētām) darbībām pāriet uz interpretējošu apstrādi (arī programmas pēc būtības izmanto meta informāciju).

Pašlaik populārākie lasāmo formātu paveidi ir XML (formālais apraksts [3] un saturīgs ievads [4]) un JSON [5]. Plašāk tiek pielietots XML, jo ir ilgstošāk populārs. Gan tāpēc, ka XML pieraksts tehniski iznāk garāks, gan organizatorisku iemeslu dēļ JSON popularitāte arvien pieaug (galvenais bremzējošais faktors ir slikta uztveramība daudziem cilvēkiem).

3.2.3. Izvēle

Izvēle starp kompaktajiem un lasāmajiem formātiem bija viegla - modeļu strukturālā sarežģītība un vēlme pēc lasāma un viegli modificējama formāta viennozīmīgi noveda pie lasāmajiem formātiem.

Izvēle starp XML un JSON vairs nebija tik acīmredzama. Gala lēmums par XML lietošanu balstījās uz sekojošiem apsvērumiem:

- lielākai daļai projekta darbinieku XML pieraksts šķita lasāmāks
- XML apstrādei ir plašāks atbalsts .NET standarta bibliotēkās
- projekta darbiniekiem ir plašāka pieredze XML izmantošanai dažādos kontekstos

Tālākajā izklāstā tiek pieņemts, ka lasītājam ir pamatzināšanas par XML.

3.3. Loģiskā struktūra

Tehniskā kodējuma izvēle nosaka tikai "kā" pierakstīt, bet atstāj ļoti plašas iespējas "ko" pierakstīt. Analoģija varētu būt ar cilvēku valodām - latviešu valoda (kodējuma veids) raksturo "kā" pierakstīt. Liekas acīmredzami, ka zem "ko" slēpjas modeļa apraksts. No "putna lidojuma" tā arī ir, bet paskatoties tuvāk arī šeit ir daudz izvēles iespēju.

3.3.1. Individuālais pret tipveida

Šī ir pamatlieta, kas jāizlemj un izlemšana nav viegla, jo atkal ir jāmeklē kompromiss. Lai būtu vieglāk saprast, par ko ir runa aplūkosim 2 XML pieraksta piemērus. Tie abi būtībā apraksta vienu un to pašu (kas ar šo piemēru domāts saturīgi, nav jāsaprot) divās dažādās pieejās - individuālajā un tipveida.

```
<server name="laacis" baseUrl="laacis:1234/MEDUS" module="./WAPc.DLL" />
<service name="Authorization" server="laacis"/>

<object type="server">
  <attribute type="name" value="laacis"/>
  <attribute type="baseUrl" value="laacis:1234/MEDUS"/>
  <attribute type="module" value="./WAPc.DLL"/>
</object>
```

```
<object type=”service”>
  <attribute type=”name” value=”Authorization”/>
  <attribute type=”server” value=””laacis”/>
</object>
```

Individuālajā pieejā mums ir 2 veidu objekti (“server” un “service”), kuriem atbilstoši ir 3 un 2 individuāli atribūti. Arī tipveida pieejā mums ir 2 veidu objekti (“object” un “attribute”), kas ir pilnīgi no cita repertuāra (arī to atribūtu komplekts ir pilnīgi cits).

Patiesībā saturīgi var teikt, ka 1. piemērs uzrakstīts mērķa vides jēdzienos, bet 2. piemērs meta līmeņa jēdzienos.

No cilvēka lasāmības viedokļa 1. piemērs ir acīmredzami labāks - stipri kompaktāks un līdz ar to arī daudz lasāmāks.

No datorapstrādes viedokļa situācija var izskatīties stipri savādāka. Katram XML dokumentam ir struktūra (formālai aprakstīšanai izmanto vai nu DTD vai XSD). Apstrādājošās programmas tiek veidotas un darbojas atbilstoši šai struktūrai. Izšķirošais moments izvēlei ir dažādo apstrādājamo struktūru (dažādo XML apstrādes programmu skaits).

Tā kā mūsu modeļi būs atbilstoši metamodeļiem, kas savukārt pakļaujas meta metamodelim, tad modeļa kodējums var būt vismaz 3 dažādos abstrakcijas līmeņos:

1. modeļa līmenis - kā XML elementi tiek izmantoti modeļa jēdzieni. Piemēram, <Person>, <server>, <konts>. Savukārt atribūtos un elementu tekstos parādās konkrētās vērtības, piemēram
   - <Person name=”Jānis Bērziņš” code=”17101936-11111”/>
2. metamodeļa līmenis - kā XML elementi tiek izmantoti metamodeļa jēdzieni. Piemēram, <Table>, <View>, <Link>. Piemēram, iepriekš apskatītais piemērs pārtop:
   - <Entity name=”Person”><Attribute name=”name” value=”Jānis Bērziņš”/> .....
3. meta metamodeļa līmenis - kā XML elementi parādās meta metaobjektu “instanču” elementi. Šie elementi atklāti nav atrodami nevienā no modelēšanas līmeņiem, bet ir samērā viegli atvasināmi no meta metamodeļa entītijām atbilstoši viņu semantikai.

Projektā plānoto metamodeļu ir ap 10, atbilstoši modeļu būtu jābūt vēl vairāk. Tas viennozīmīgi izslēdz modeļa līmeni kā kodēšanas pamatu.

3.3.2. Maināmība

Par maināmību jārunā divos dažādos aspektos. Vienkāršākais ir gatavu modeļu XML failu labošana - tehniski XML fails ir labojams ar jebkuru teksta redaktoru. Šeit vislielākās grūtības sagādā objektu savstarpējās saites, kas izmaiņu procesu var padarīt ļoti sarežģītu.

Būtiskāka ir metainformācijas (metamodeļu un meta metamodeļa) mainīšanās (patiesībā visa šī projekta pamatmērķis ir spēt labi tikt galā ar visu laiku mainīgajām prasībām). Tas savukārt noved pie **kļūdu tolerances** - tikt galā ar modeļu aprakstiem, kas neatbilst metamodeļa aprakstam (varbūt pat meta metamodeļa). Neatbilstošie elementi būtu ignorējami (varbūt pēc kartējām metamodeļa izmaiņām tie atkal kļūs derīgi).

Lielā mērā loģiskās struktūras izstrādes principus kā atbaidošs piemērs iespaidoja Microsoft DSL [6] modeļu apraksta formāts (individuāls ar tolerances trūkumu).

Augstāk minēto apsvērumu rezultātā tika izlemts veidot *uz meta metamodeļa instancēšanas bāzētu tolerantu formātu*, t.i. patiesībā formātu, kas ir vēl plašāks nekā meta metamodeļa vajadzībām nepieciešams.

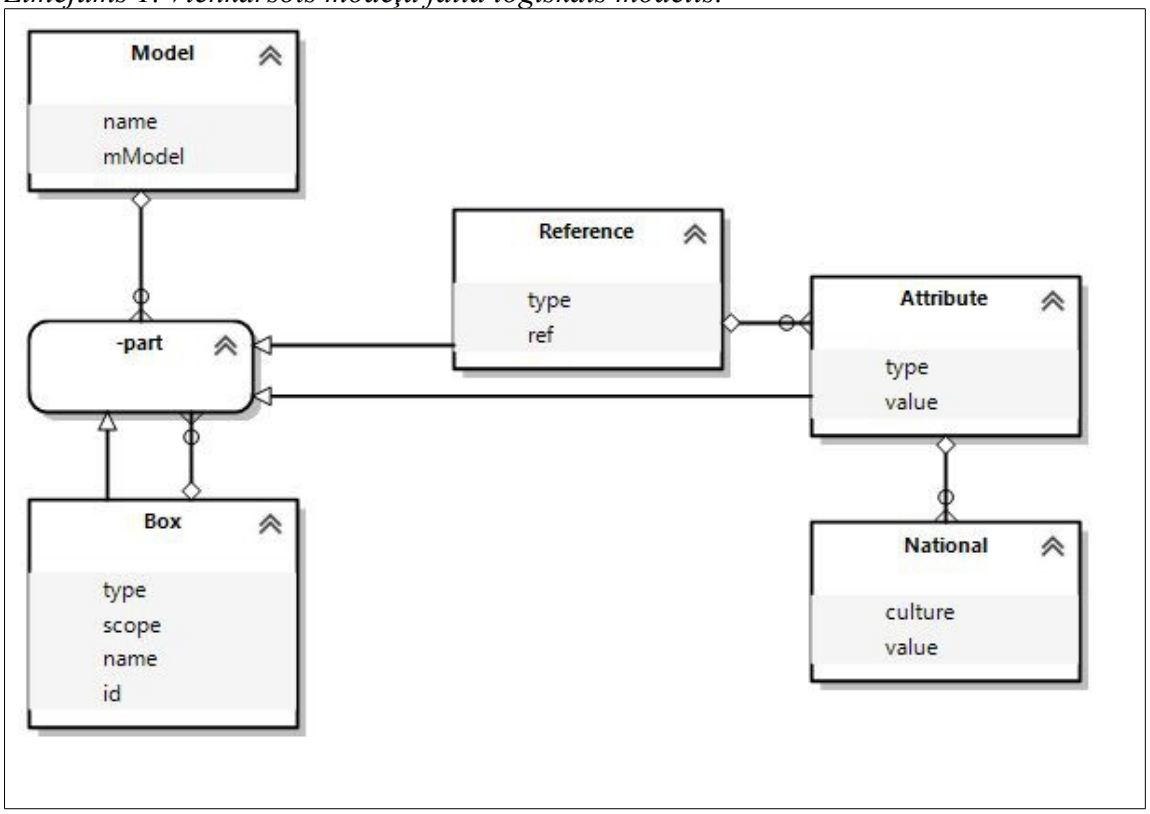
4. Formāta definīcija

Precīza formāla formāta definīcija dota pielikumos (gan DTD formā gan XSD formā). Šī nodaļa veltīta saturīgajam sastāvdaļu izskaidrojumam. Saturīgi formāts apraksta atributētus objektus ar atributētām norādēm uz objektiem. Atribūti ir ar daudzvalodības atbalstu, t.i. atribūta vērtības var tikt uzdotas paralēli vairākās valodās (vai precīzāk runājot, kultūrās[7]).

4.1. Pamatstruktūra

Zīmējumā zemāk attēlota nedaudz vienkāršota (vēlāk tiks izskaidrotas šeit noslēptās lietas) modeļa faila struktūra (lietotie apzīmējumi izskaidroti [1]).

Zīmējums 1. Vienkāršots modeļu faila loģiskais modelis.



Visas modeļa entītijas tiek attēlotas kā XML elementi (arī mulsinošā entītija "Attribute" attēlojas par XML elementu), visi modeļa atribūti - XML elementu atribūti.

4.1.1. Model

Modeļa elements, kas ir arī XML koka sakne, apraksta modeli kopumā. Tam var būt savi atribūti (Attribute) un atsauces uz objektiem (Reference). Galvenais, protams, ir tas, ka modelis satur objektus (Box), kas var būt hierarhiski iekļauti.

Individuāli modelim ir sekojošie atribūti (XML atribūti):

- name modeļa vārds saturīgai identifikācijai;
- mModel metamodeļa, kuram atbilst šis modelis, vārds. Formātā šim vārdam nav saturīgas nozīmes.

4.1.2. Box

Modeļa elements apraksta modeļa objektus. Tam var būt savi atribūti (Attribute), atsauces uz objektiem (Reference) un hierarhiski pakļautus citus objektus.

Individuāli šim elementam ir sekojošie atribūti (XML atribūti):

- name objekta vārds saturīgai identifikācijai;
- type objekta tips - saturīgi modeļa apraksta failā neko nenozīmē, bet tipskos pielietojumos varētu būt atbilstošā metamodeļa objekta vārds;
- scope papildus vārds, kas kopā ar "name" varētu tikt izmantots unikālu salikto vārdu būvēšanai (analogs C# namespace), bet modeļa apraksta failā neko nenozīmē;
- id unikāls objekta (Box) identifikators šī modeļa apraksta faila ietvaros. Struktūra nekādi netiek reglamentēta.

4.1.3. Reference

Modeļa elements apraksta atsauci uz modeļa objektu. Tam var būt savi atribūti (Attribute).

Individuāli šim elementam ir sekojošie atribūti (XML atribūti):

- type atsauces tips - saturīgi modeļa apraksta failā neko nenozīmē, bet tipskos pielietojumos varētu būt atbilstošā metamodeļa objekta vārds;
- ref unikālais objekta (Box), uz kuru šī atsauce norāda, identifikators (id). Atsauces darbojas tikai viena modeļa apraksta faila ietvaros.

4.1.4. Attribute

Modeļa elements apraksta saturošā (XML nozīmē) objekta (modelis, objekts, atsauce) atribūta vērtību pamata kultūrā. Tas var saturēt vērtības arī citās kultūrās, izmantojot "National" elementus.

Individuāli šim elementam ir sekojošie atribūti (XML atribūti):

- type atribūta tips - saturīgi modeļa apraksta failā neko nenozīmē, bet tipskos pielietojumos varētu būt atbilstošā metamodeļa atribūta vārds;
- value atribūta vērtība (pamata kultūrā).

4.1.5. National

Modeļa elements apraksta saturošā atribūta vērtību kādā papildus kultūrā.

Individuāli šim elementam ir sekojošie atribūti (XML atribūti):

- culture kultūras identifikators. Viena atribūta ietvaros kultūras identifikatoram jābūt unikālam (nedrīkst vienai kultūrai paredzēt dažādas vērtības). Kultūras identifikatora vērtības struktūra šī apraksta ietvaros nav precizēta, bet visdabiskākais saturs varētu būt atbilstošs Microsoft CultureInfo definīcijai [8].
- value atribūta vērtība.

4.2. Paplašinājumi

Paplašinājumi dod papildus iespējas, kas nemaina saturīgo uzbūvi.

4.2.1. Atribūts “comment”

Visiem iepriekš aprakstītajiem elementiem ir vēl papildus atribūts (XML nozīmē) *comment*, kas var saturēt patvaļīgu tekstu. Kā rāda nosaukums atribūts domāts komentāra pievienošanai.

4.2.2. Comment

Šis elements var tikt iekļauts jebkurā no aprakstītajiem elementiem un domāts komentāru pievienošanai. Tas satur vienu atribūtu “comment”. Tas, vai izvēlēties komentāra atribūtu vai komentāra elementu, pilnīgi atkarīgs no estētiskiem apsvērumiem.

4.2.3. Message

Šis elements var tikt iekļauts jebkurā no iepriekš aprakstītajiem elementiem, izņemot komentāra elementu, un ir veids, kā apstrādājošai programmai atstāt savas piezīmes par modeļa elementiem (piemēram, paziņojumus par pamanītām kļūdām).

Individuāli šim elementam ir sekojošie atribūti (XML atribūti), kuru aizpildīšana un interpretācija ir pilnīgi apstrādājošo programmu ziņā:

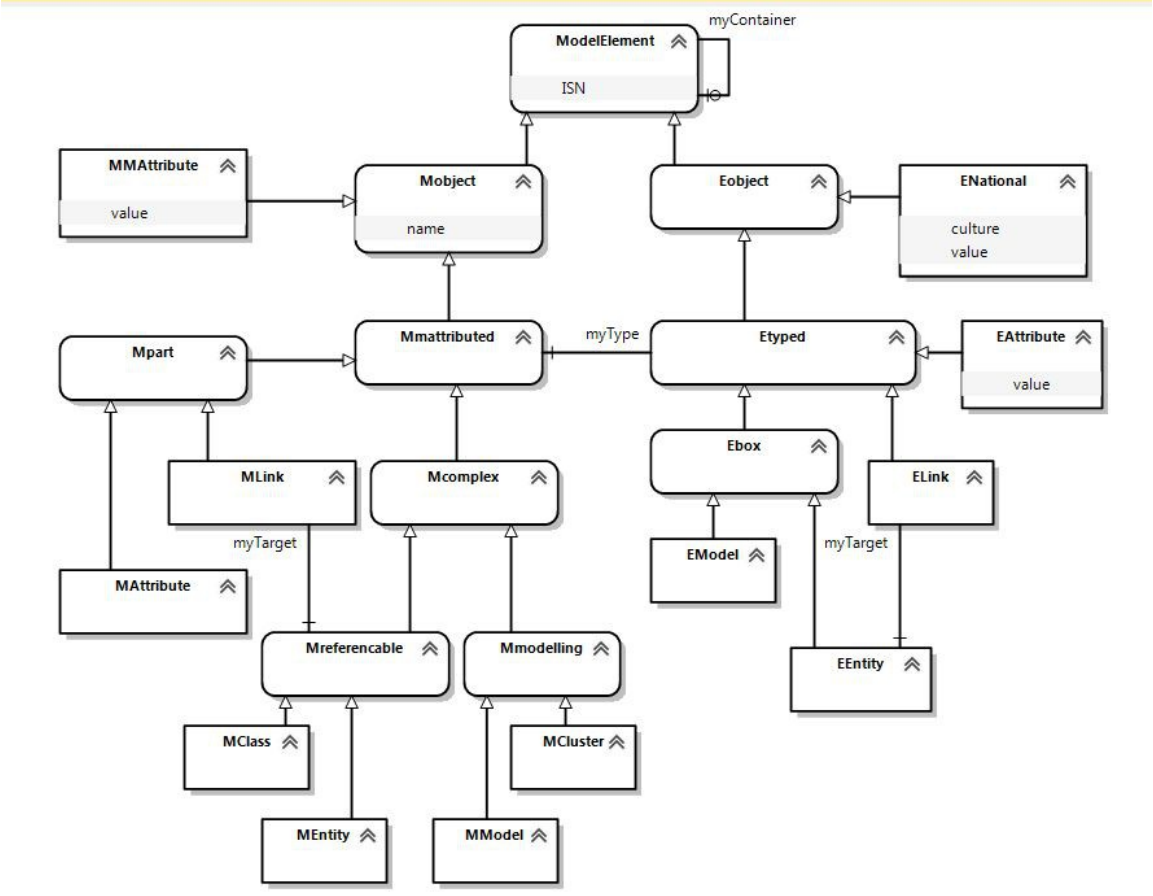
- type ziņojuma veids. Ir atļautas vērtības “E” (kļūda) un “W” (brīdinajums).
- mesid ziņojuma identifikators.
- text ziņojuma teksts.

5. Modeļu kodēšana

Iepriekšējās nodaļas aprakstīja apsvērumus formāta izveidei un gatavo formāta definīciju. Tā kā formāta definīcija ir vispārīgāka par to kā tiek definēti modeļi (meta metamodeļa un metamodeļa definīcijas uzliek papildus ierobežojumus) ir izveidojusies situācija, ka modeli šajā formātā var iekodēt dažādos veidos. Šī nodaļa satur ir konkrētā kodēšanas veida aprakstu.

Lai būtu vieglāk saprast kodēšanas veidu aplūkosim kombinēta metamodeļa un modeļa glabāšanas modeli (būtībā klašu hierarhija apstrādes sistēmā).

Zīmējums 2. Metamodeļa un modeļa glabāšanas modelis.



Kreisā puse (Mobject) patiesībā saturīgi atkārto meta metamodeli un tiek izmantota metamodeļa glabāšanai. Savukārt, labā puse (Eobject) satur modeļa informāciju un ir šīs nodaļas primārais interešu objekts. Patiesībā modeļa apraksta fails ir šīs daļas kodējums ar nelielu piedevu no kreisās puses.

Pielikumā 8.3 ir atrodams reāla modeļa XML piemērs, kas varētu palīdzēt vieglak saprast sekojošo aprakstu.

5.1. myContainer

Atsauce *myContainer* tehniski atrodas pie *ModelElement*, bet saturīgi tā, protams, ir visiem atbilstošajiem *Eobject* paveidiem un norāda savstarpējo iekļautību. Šī atsauce tiek modeļa apraksta failā kodēta kā atbilstošo elementu iekļaušana.

5.2. EModel

Šis elements var būt tikai tieši vienā eksemplārā un apraksta modeli. Tam atbilstošs ir XML elements *Model*. Atribūts *name* netiek aizpildīts. Savukārt, *mModel* tiek aizpildīts no kreisās puses - arī *MModel* ir tieši vienā eksemplārā un tā *name* vērtība nonāk XML atribūtā.

5.3. myType

Atsauce *myType* tehniski atrodas pie *Etyped*, bet saturīgi tā, protams, ir viesiem atbilstošajiem *Etyped* paveidiem un norāda uz atbilstošu (piemēram, no *EAttribute* ir norāde *myType* uz *MAttribute* instanci) metamodeļa objekta instanci. Šī mērķa objekta *name* vērtība dod XML atribūta *type* vērtību.

5.4. EEntity

Šis objekts tie kodēts kā *Box* elements XML formātā. Tā un arī tā sastāvdaļu novietojumu XML kokā pilnīgi nosaka *myContainer* atsauces, kā aprakstīts iepriekš. *Box* XML atribūti tiek aizpildīti sekojoši:

- *id* mehāniski ģenerēts unikāls identifikators;
- *name* tiek aizpildīts tikai, lai XML attēlojums būtu lasāmāks. Šeit tiek iepildīts *EEntity* saturīgais vārds, ko meklē pēc sekojoša algoritma:
    - caur *myType* metamodelī tiek atrasts atbilstošais *MEntity;*
    - metainformācijā tiek atrasts šim *MEntity* piederošs *MAttribute*, kura meta metaatribūta *isName* vērtība ir *true*;
    - modeļa pusē tiek atrasts aplūkojamajam *EEntity* piederošais *EAttribute*, kura *myType* rāda uz iepriekšējā solī atrasto *MAttribute* un lauka *value* vērtība nonāk XML atribūtā *name;*
- *scope* netiek aizpildīts*;*
- *type* tiek aizpildīts ar *MEntity* vārdu, kā iepriekš aprakstīts par *myType.*

5.5. EAttribute

Šis objekts tiek kodēts kā XML elements *Attribute* atbilsoši *myContainer* un *myType* (rāda uz *MAttribute*) iepriekš aprakstītajai loģikai. Vienīgais speciālais lauks ir *value*, kurš nonāk XML atribūtā *value.*

5.6. ENational

Šis objekts tiek kodēts kā XML elements *National* ar novietojumu atbilstoši *myContainer* (rāda uz *EAttribute*) iepriekš aprakstītajai loģikai. Lauki *culture* un *value* nonāk atbilstošajos XML atribūtos.

5.7. ELink

Šis objekts tiek kodēts kā XML elements *Reference* atbilsoši *myContainer* un *myType* (rāda uz *MLink*) iepriekš aprakstītajai loģikai. Vienīgais speciālais lauks ir norāde *myTarget*, kurš kļūst par XML atribūtu *ref.* Tā vērtība ir *myTarget* mērķa *EEntity* atbilstošā *Box* atribūta *id* vērtība.

6. Secinājumi un rezultāti

Aktivitātes ietvaros veiktā darba rezultātā izstrādāts modeļu apmaiņas formāts, kas ir viegli lasāms, modificējams un pie kļūdu tolerants, tas ir, tiek galā ar modeļu aprakstiem, kas neatbilst metamodeļa aprakstam (varbūt pat meta metamodeļa). Tā kā izstrādātais formāts ir vēl plašāks nekā meta metamodeļa vajadzībām nepieciešams un modeli šajā formātā var iekodēt dažādos veidos, tad tika izstrādāts konkrēts modeļu apmaiņas formāta kodēšanas veids.

Šīs aktivitātes iegūtie rezultāti tiks tālāk izmantoti Modeļa ietvara izstrādē.

7. Literatūras saraksts

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

[2] http://en.wikipedia.org/wiki/Comma-separated_values

[3]http://www.w3.org/TR/REC-xml/

[4]http://en.wikipedia.org/wiki/Xml

[5]http://en.wikipedia.org/wiki/Json

[6]http://msdn.microsoft.com/en-us/library/bb126327.aspx

[7]http://www.csharp-examples.net/culture-names/

[8]http://msdn.microsoft.com/en-us/library/vstudio/system.globalization.cultureinfo

8. Pielikumi

8.1. Formāta definīcijas DTD

```
<!ELEMENT Model             (Attribute | Box | Comment | Message | Reference)* >

<!ELEMENT Attribute   (National | Comment | Message)* >
<!ELEMENT Box      (Attribute | Box | Comment | Message | Reference)* >
<!ELEMENT Comment     EMPTY >
<!ELEMENT Message     EMPTY >
<!ELEMENT National    (Comment | Message)* >
<!ELEMENT Reference   (Attribute | Comment | Message)* >

<!ATTLIST Attribute  type   CDATA #REQUIRED
                            value  CDATA #IMPLIED
                            comment CDATA #IMPLIED >
<!ATTLIST Box        type   CDATA #REQUIRED
                     scope   CDATA #IMPLIED
                     name    CDATA #REQUIRED
                     id      ID    #REQUIRED
                            comment CDATA #IMPLIED >
<!ATTLIST Comment    comment CDATA #IMPLIED >
<!ATTLIST Message    type    (E|W) "E"
                     mesid   CDATA #IMPLIED
                     text    CDATA #IMPLIED  >
<!ATTLIST Model          name    CDATA #IMPLIED
                     mModel  CDATA #IMPLIED
                            comment CDATA #IMPLIED >
<!ATTLIST National   culture CDATA #REQUIRED
                            value   CDATA #REQUIRED
                            comment CDATA #IMPLIED >
<!ATTLIST Reference type    CDATA #REQUIRED
                     ref     IDREF #IMPLIED
                            comment CDATA #IMPLIED >
```

8.2. Formāta definīcijas XSD

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="Model">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="Attribute"/>
        <xs:element ref="Box"/>
        <xs:element ref="Comment"/>
        <xs:element ref="Message"/>
        <xs:element ref="Reference"/>
      </xs:choice>
      <xs:attribute name="name"/>
      <xs:attribute name="mModel"/>
      <xs:attribute name="comment"/>
    </xs:complexType>
  </xs:element>
```

```
<xs:element name="Attribute">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="National"/>
      <xs:element ref="Comment"/>
      <xs:element ref="Message"/>
    </xs:choice>
    <xs:attribute name="type" use="required"/>
    <xs:attribute name="value"/>
    <xs:attribute name="comment"/>
  </xs:complexType>
</xs:element>
<xs:element name="Box">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="Attribute"/>
      <xs:element ref="Box"/>
      <xs:element ref="Comment"/>
      <xs:element ref="Message"/>
      <xs:element ref="Reference"/>
    </xs:choice>
    <xs:attribute name="type" use="required"/>
    <xs:attribute name="scope"/>
    <xs:attribute name="name" use="required"/>
    <xs:attribute name="id" use="required" type="xs:ID"/>
    <xs:attribute name="comment"/>
  </xs:complexType>
</xs:element>
<xs:element name="Comment">
  <xs:complexType>
    <xs:attribute name="comment"/>
  </xs:complexType>
</xs:element>
<xs:element name="Message">
  <xs:complexType>
    <xs:attribute name="type" default="E">
      <xs:simpleType>
        <xs:restriction base="xs:token">
          <xs:enumeration value="E"/>
          <xs:enumeration value="W"/>
        </xs:restriction>
      </xs:simpleType>
    </xs:attribute>
    <xs:attribute name="mesid"/>
    <xs:attribute name="text"/>
  </xs:complexType>
</xs:element>
<xs:element name="National">
  <xs:complexType>
    <xs:choice minOccurs="0" maxOccurs="unbounded">
      <xs:element ref="Comment"/>
      <xs:element ref="Message"/>
    </xs:choice>
    <xs:attribute name="culture" use="required"/>
    <xs:attribute name="value" use="required"/>
    <xs:attribute name="comment"/>
  </xs:complexType>
</xs:element>
```

```
  <xs:element name="Reference">
    <xs:complexType>
      <xs:choice minOccurs="0" maxOccurs="unbounded">
        <xs:element ref="Attribute"/>
        <xs:element ref="Comment"/>
        <xs:element ref="Message"/>
      </xs:choice>
      <xs:attribute name="type" use="required"/>
      <xs:attribute name="ref" type="xs:IDREF"/>
      <xs:attribute name="comment"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

8.3. Modeļa faila piemērs

```
<?xml version="1.0" encoding="utf-8"?>
<Model xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" mModel="Entities">
 <Attribute type="dataBase" value="Localization" />
 <Box type="Entity" name="LocalizationSupportCulture" id="53">
  <Attribute type="name" value="LocalizationSupportCulture" />
  <Box type="Unit" name="clName" id="55">
   <Attribute type="name" value="clName" />
   <Attribute type="cType" value="string" />
  </Box>
  <Box type="Unit" name="id" id="58">
   <Attribute type="name" value="id" />
   <Attribute type="cType" value="int" />
  </Box>
 </Box>
 <Box type="Entity" name="LocalizationSupportGroup" id="61">
  <Attribute type="name" value="LocalizationSupportGroup" />
  <Box type="Unit" name="grName" id="63">
   <Attribute type="name" value="grName" />
   <Attribute type="cType" value="string" />
  </Box>
  <Box type="Unit" name="id" id="66">
   <Attribute type="name" value="id" />
   <Attribute type="cType" value="int" />
  </Box>
 </Box>
 <Box type="Entity" name="LocalizationSupportElement" id="69">
  <Attribute type="name" value="LocalizationSupportElement" />
  <Box type="Unit" name="elName" id="71">
   <Attribute type="name" value="elName" />
   <Attribute type="cType" value="string" />
  </Box>
```

```
    <Box type="Unit" name="inGroup" id="74">
      <Attribute type="name" value="inGroup" />
      <Attribute type="cType" value="int?" />
    </Box>
    <Box type="Unit" name="id" id="77">
      <Attribute type="name" value="id" />
      <Attribute type="cType" value="int" />
    </Box>
  </Box>
  <Box type="Entity" name="LocalizationSupportValue" id="80">
    <Attribute type="name" value="LocalizationSupportValue" />
    <Box type="Unit" name="value" id="82">
      <Attribute type="name" value="value" />
      <Attribute type="cType" value="string" />
    </Box>
    <Box type="Unit" name="ofElement" id="85">
      <Attribute type="name" value="ofElement" />
      <Attribute type="cType" value="int?" />
    </Box>
    <Box type="Unit" name="inCulture" id="88">
      <Attribute type="name" value="inCulture" />
      <Attribute type="cType" value="int?" />
    </Box>
    <Box type="Unit" name="isManuallyEdited" id="91">
      <Attribute type="name" value="isManuallyEdited" />
      <Attribute type="cType" value="bool?" />
    </Box>
    <Box type="Unit" name="id" id="94">
      <Attribute type="name" value="id" />
      <Attribute type="cType" value="int" />
    </Box>
  </Box>
</Model>
```