LATVIJAS UNIVERSITĀTE
Matemātikas un informātikas institūts

EDGARS CELMS

TRANSFORMĀCIJU VALODA MOLA UN TĀS LIETOJUMI

Promocijas darba kopsavilkums
datorzinātņu doktora (Dr. sc. comp.) zinātniskā grāda iegūšanai

Nozare: datorzinātnes
Apakšnozare: programmēšanas valodas un sistēmas

Zinātniskais vadītājs:
profesors, Dr. habil. sc. comp.
AUDRIS KALNIŅŠ

Rīga - 2007

Darbs ir izstrādāts ar Eiropas Sociālā fonda atbalstu. Projekts „Doktorantu un jauno zinātnieku pētniecības darba atbalsts Latvijas Universitātē”

Darba vadītājs:

Profesors, Dr. habil. sc. comp. Audris Kalniņš
Latvijas Universitāte

Recenzenti:

Profesors, Dr. sc. comp. Jānis Bičevskis
Latvijas Universitāte

Asoc. Profesore, Dr. sc. ing. Mārīte Kirikova
Rīgas Tehniskā universitāte

Dr. sc. comp. Uģis Sarkans
Eiropas Bioinformātikas institūts (Kembridža, Lielbritānija)

Darba aizstāvēšana notiks Latvijas Universitātes Datorzinātnes nozares promocijas padomes atklātā sēdē 11.09.2007 LU Matemātikas un informātikas institūtā (Raiņa bulvārī 29), 413. auditorijā.

Ar darbu un tā kopsavilkumu var iepazīties LU bibliotēkā (Kalpaka bulvārī 4).

Padomes priekšsēdētājs Jānis Bārzdiņš

Saturs

Promocijas darba tēmas aktualitāte un iegūtie rezultāti 4
Promocijas darba vispārējais raksturojums 6
1 Metamodelēšana un modeļu balstīta sistēmu izstrāde 8
1.1 Metamodelēšana, metamodeļi un modeļi 8
1.2 Modeļu balstīta sistēmu izstrāde 10
1.3 Modeļu transformāciju valodas 12
2 Redaktoru definēšanas valoda un universālais modelēšanas rīks 14
2.1 Redaktoru definēšanas valoda un universālais modelēšanas rīka prototips. 14
2.2 Universālais modelēšanas rīks 17
3 Modeļu transformāciju valoda MOLA un tās lietošanas metodoloģija 22
3.1 Modeļu transformāciju valoda MOLA 22
3.2 Valodas MOLA lietošanas metodoloģija un paraugu atpazīšanas efektivitāte 25
4 Valodas MOLA implementācija un valodas lietojumi 27
5 Nobeigums 29
6 Atsauces 30
6.1 Autora publikācijas recenzētos starptautisku konferenču materiālos 30
6.2 Citas autora publikācijas recenzētos starptautisku konferenču materiālos (tieši nesaistītas ar promocijas darba tēmu) 30
6.3 Citi promocijas darbā izmantotie avoti 31
7 Pielikums 33
7.1 Autora referāti par darba rezultātiem starptautiskās zinātniskās konferencēs vai semināros 33
7.2 Promocijas darbā iekļautās publikācijas un promocijas darba autora personiskais ieguldījums. 33

Promocijas darba tēmas aktualitāte un iegūtie rezultāti

Problēmas aktualitāte:
Mūsdienu praksē specifiskos problēmu apgabalos tiek izstrādātas un lietotas specializētas modelēšanas valodas – domēnu specifiskas valodas[16] (DSL – *Domain Specific Language*), kuras ir paredzētas kādas konkrētas uzdevumu grupas veikšanai. Šādu valodu atbalsta rīku izstrādāšana ir darbietilpīgs process. Viens no veidiem, kā uzlabot DSL rīku izstrādes procesu, ir veidot universālas metamodeļbāzētas rīku izstrādes platformas, kuras nodrošinātu ērtu un ātru atbilstošo rīku izstrādi. Daļa no promocijas darba pētījumiem ir veltīti tieši universālu metamodeļbāzētu rīku izstrādāšanas problemātikai.

Cits aktuāls problēmu apgabals, kuram ir veltīti promocijas darba pētījumi, ir saistīts ar modeļu transformāciju valodām. Modeļu transformāciju valodas ir kļuvušas par netriviālu datorsistēmu izstrādes procesa būtisku sastāvdaļu. Mūsdienās, būvējot datorsistēmas, to izstrādē arvien biežāk tiek izmantoti dažādi metamodeļi un tiem atbilstošie modeļi. Tie ieņem arvien nozīmīgāku vietu sistēmu būvē. Šāds izstrādes veids tiek saukts par modeļu balstītu sistēmu izstrādi (*MDSD – Model Driven Software Development*)[17, 18]. Tā ir tehnoloģija, kura strauji attīstās un tiek arvien plašāk pielietota dažādu sistēmu projektēšanā. Jāpiezīmē, ka nereti MDSD pieeja sistēmu izstrādē balstās uz dažādām DSL valodām, ar kuru palīdzību tiek veidoti atbilstošie modeļi. Pamatideja, pielietojot MDSD pieeju sistēmu būvē, ir izmantot modeļus, kurus ir iespējams noteiktos projektēšanas posmos pārveidot vienu par otru, pielietojot atbilstošas transformācijas. Līdz ar to ļoti svarīgi ir līdzekļi, ar kuriem šādas modeļu transformācijas var veikt – modeļu transformāciju valodas. Lietojot MDSD pieeju, ir nepieciešamas programmēšanas valodas, kuras būtu paredzētas modeļus transformējošu procesu aprakstīšanai. Praksē ir pierādījies, ka neviena no esošajām tradicionālajām programmēšanas vai modelēšanas valodām nav pietiekami piemērota tām prasībām, kuras ir nepieciešamas modeļu transformāciju aprakstīšanā, un, lai varētu realizēt MDSD idejas praksē, ir nepieciešama speciāla veida, konkrēti šiem mērķiem paredzēta valoda. Promocijas darbā izstrādātā valoda MOLA ir tieši šāda veida valoda, kura ir ērti lietojama MDSD uzdevumos. Jāpiezīmē, ka, neskatoties uz jau vairākus gadus veiktajiem plašajiem pētījumiem par modeļu transformāciju valodām, pasaulē vadošā standartus izstrādājošā organizācija modelēšanas jomā (OMG[19]) vēl arvien nav izstrādājusi modeļu transformāciju valodas standartu.

Jaunākie pētījumi rāda, ka transformāciju valodas ir arī ērti lietojamas DSL rīku būvē. Jāpiezīmē, ka arī promocijas darbā izstrādātā transformāciju valoda MOLA ir tipisks DSL valodas piemērs.

Darba galvenie rezultāti:

- Izstrādāts **universāls metamodeļbāzēts modelēšanas rīks** – pēc būtības tā ir pilnībā metamodeļbāzēta modelēšanas rīku būves platforma. Galvenā rīka priekšrocība un īpatnība ir tā, ka tam nav predefinētas (iebūvētas) modelēšanas metodoloģijas. Lai sāktu kaut ko modelēt, rīks sākumā ir "jāuzpilda" ar metamodeli un papildus informāciju, kura apraksta vēlamo modelēšanas valodu un atbilstošo tehnoloģiju darbam ar modeļiem. Rezultātā ar relatīvi nelielu darba ieguldījumu var izveidot rūpnieciskas kvalitātes konkrētu

metodoloģiju atbalstošu modelēšanas rīku. Jāuzsver, ka šis rīks tika sekmīgi izmantots arī tālākos pētījumos un, izmantojot šo rīku, tika izveidots modeļu transformāciju valodas MOLA redaktors.

- Izstrādāta **modeļu transformāciju valoda MOLA** – grafiska modeļu transformāciju valoda, kura ir jau pārbaudīta praksē, ir ērti lietojama un ļoti piemērota valoda MDSD uzdevumiem.
- Izstrādāts **MOLA rīks** – tas ir vairāku komponenšu apvienojums, kas nodrošina visus nepieciešamos servisus, lai valodā MOLA rakstītas transformācijas varētu pielietot MDSD uzdevumos. Rīks nodrošina tādus servisus kā transformāciju rakstīšana, kompilēšana, izpilde, datu imports un eksports no/uz rūpnieciskiem modelēšanas rīkiem.

Jāpiezīmē, ka iegūtie rezultāti ir kolektīva darba rezultāts. Promocijas darba autors ir devis būtisku ieguldījumu un aktīvi piedalījies visos pētījumos, kuri ir saistīti ar iegūtajiem rezultātiem. Autora personiskais ieguldījums ir konkretizēts turpmākajās kopsavilkuma nodaļās un pielikumā.

Promocijas darba vispārējais raksturojums

Promocijas darbs "**Transformāciju valoda MOLA un tās lietojumi**" ir izstrādāts no 2001. līdz 2007. gadam Latvijas Universitātes Fizikas un matemātikas fakultātē un Latvijas Universitātes Matemātikas un informātikas institūtā (MII) profesora Audra Kalniņa vadībā. Šis darbs turpina MII rīku būves un valodu izstrādes tradīcijas, kas aizsāktas jau 1986. gadā. Promocijas darba pētījumu galvenie rezultāti ir atspoguļoti 13 publikācijās [1-13]. Promocijas darbs ir **tematiski vienota publikāciju kopa**, kurā ir atspoguļoti autora pētījumu rezultāti dažādos ar metamodeļbāzētu modelēšanas rīku būvi un modeļu transformāciju valodām saistītos aspektos.

Pētījuma priekšmets – universālu metamodeļbāzētu modelēšanas rīku būve un modeļu transformāciju valodas.

Pētījuma mērķis – izstrādāt un ieviest praksē universālu metamodeļbāzētu modelēšanas rīku, kā arī izstrādāt un eksperimentāli pārbaudīt modeļu transformāciju valodu, kura būtu pielietojama MDSD uzdevumos.

Pētījuma posmi – promocijas darba rezultāti veido noslēgtu pētījumu ciklu, sākot no pētījumiem, kuru viens no galvenajiem rezultātiem ir izstrādātais universālais modelēšanas rīks (rīks vēlāk tika izmantots modeļu transformāciju valodas izstrādē) un beidzot ar pētījumiem par konkrētu modeļu transformāciju valodu MOLA, tās implementēšanu un pielietošanu MDSD uzdevumos. Promocijas darba ietvaros veiktie pētījumi ir apkopoti tematiskā kārtībā (tas atbilst arī pētījumu veikšanas hronoloģiskai secībai):

- **Pirmais pētījumu posms** – no 2000. līdz 2004. gadam. Šī posma pētījumi un rezultāti ir aprakstīti kopsavilkuma 2. nodaļā un atbilstošajās publikācijās [1-5]. Galvenais šī posma pētījumu rezultāts ir izstrādātais **universālais metamodeļbāzētais modelēšanas rīks**, kurš daudzos aspektos tajā laikā bija unikāls un pārāks par citiem rīkiem, kuri pretendēja uz līdzīgu redaktoru statusu. Otrs nozīmīgs rezultāts bija iegūtā izpratne un zināšanas par modeļu transformācijām. Šajā laika posmā iegūtie pētījumu rezultāti bija galvenais ideju avots turpmākajai modeļu transformāciju valodas MOLA izstrādei un ar to saistītajiem pētījumiem. Promocijas darba autors ir aktīvi piedalījies visos pētījumos šajā laika posmā, sākot ar 2001. gadu. Autora ieguldījums kopumā sastāda vairāk kā 40% no visiem veiktajiem pētījumiem šajā laika posmā.

- **Otrais pētījumu posms** – no 2003. līdz 2006. gadam. Šī posma pētījumi un rezultāti ir aprakstīti kopsavilkuma 3. nodaļā un atbilstošajās publikācijās [5-10]. Galvenais pētījumu rezultāts ir izstrādātā **modeļu transformāciju valoda MOLA.** Jāpiezīmē, ka pētījumi šajā jomā tiek turpināti un pašlaik tiek izstrādāta nākamā valodas versija. Nozīmīgs pētījumu rezultāts ir arī veiktie valodas MOLA paraugu atpazīšanas efektivitātes novērtējumi. Promocijas darba autors ir aktīvi piedalījies visos pētījumos, kuri tika veikti šajā laika posmā un autora personiskais ieguldījums kopumā sastāda vairāk kā 50% no visiem veiktajiem pētījumiem šajā posmā. Jāuzsver, ka šajos pētījumos izstrādātajiem risinājumiem vēlāk bija ļoti liela praktiska nozīme efektīvai valodas MOLA implementācijai.

- **Trešais pētījumu posms** – no 2005. līdz 2007. gadam. Šī posma pētījumi un rezultāti ir aprakstīti kopsavilkuma 4. nodaļā un atbilstošajās publikācijās [11-13]. Galvenais pētījumu rezultāts ir izstrādātais **valodas MOLA rīks**. Promocijas darba autors ir aktīvi piedalījies visos pētījumos, kuri ir veikti šajā laika posmā un autora personiskais ieguldījums kopumā sastāda vairāk kā 30% no visiem veiktajiem pētījumiem šajā posmā. Jāpiezīmē, ka pētījumi arī šajā jomā tiek intensīvi turpināti un pašlaik notiek darbs pie nākamās valodas MOLA rīka versijas.

Pētījumu teorētiskā un praktiskā nozīme.
Promocijas darba pētījumu galvenie rezultāti ir atspoguļoti 13 publikācijās [1-13]. Par darba galvenajiem rezultātiem ir ziņots pētījumu laika posma nozīmīgākajās starptautiskās zinātniskajās konferencēs par doto tematiku, no kurām septiņās ir referējis promocijas darba autors. Promocijas darba ietvaros izstrādāto MOLA rīku (*MOLA Tool*) autors ir demonstrējis starptautiskas konferences ("*European Conference on Model Driven Architecture – Foundations and Applications (ECMDA-FA)*") rīku demonstrācijas sekcijā [20].

Uzskatāms pierādījums pētījumu rezultātu **praktiskai nozīmei** ir izstrādātās modeļu transformāciju valodas MOLA izmantošana dažādos projektos. Zemāk ir minēti interesantākie no tiem:
- Eiropas Savienības 6. Ietvara projekts – ReDSeeDS (*Requirements-Driven Software Development System*) [21]. ReDSeeDS projektā valoda MOLA tiek izmantota kā transformāciju definēšanas un izpildes līdzeklis.
- Šobrīd MII tiek izstrādāta jaunas paaudzes uz metamodeļiem un modeļu transformācijām balstīta rīku platforma (*Transformation-based Tool Framework*) [22], kurā modeļu transformācijas tiek būvētas MOLA valodā.
- LU datorzinātņu maģistrantūrā ir izstrādāts un ieviests kurss „MDA un modeļu transformācijas", kurā plaši tiek izmantota valoda MOLA gan studentu praktiskajos darbos, gan arī modeļu balstītu sistēmas izstrādes pamatprincipu izklāstā.

Kopsavilkuma tālākajās nodaļās ir īsi izklāstīti pētījumu būtiskākie rezultāti un galvenās problēmu nostādnes, kas atbilstošajās publikācijās ir aprakstītas detalizētāk:
- Pirmajā nodaļā ir dots vispārējs pārskats par modeļu balstītu sistēmu izstrādes pamatprincipiem un galvenajiem terminiem, tādejādi sniedzot lasītājam pamatzināšanas, kas nepieciešamas labākai autora veikto pētījumu un iegūto rezultātu nozīmīguma izpratnei.
- No otrās līdz ceturtajai nodaļai ir dots konspektīvs veikto pētījumu un iegūto rezultātu apraksts.
- Nobeigumā ir dots konspektīvs pētījumu rezultātu apkopojums un īsi aprakstīti turpmākie pētījumu virzieni.
- Pielikumā ir uzskaitītas starptautiskās zinātniskās konferences, kurās autors ir referējis veikto pētījumu rezultātus un konkretizēts autora personiskais ieguldījums katrā no promocijas darbā iekļautajām publikācijām.

1 Metamodelēšana un modeļu balstīta sistēmu izstrāde

Mūsdienās, būvējot netriviālas datorsistēmas, to izstrādē arvien biežāk tiek izmantoti dažādi metamodeļi un tiem atbilstošie modeļi. Tie ieņem arvien nozīmīgāku lomu sistēmu būvē. Sistēmu izstrādes procesā ar šiem modeļiem tiek veiktas dažādas transformācijas (pārveidojumi). Savukārt, lai aprakstītu transformācijas tiek lietotas transformāciju valodas. Šāds izstrādes veids tiek saukts par modeļu balstītu sistēmu izstrādi (*MDSD – Model Driven Software Development*)[17, 18]. Nodaļā tiek dots vispārējs pārskats par šo datorzinātnes nozari un tās pielietošanu sistēmu būvē, par galvenajiem terminiem šajā nozarē, tādejādi nodrošinot lasītāju ar pamatzināšanām, kas nepieciešamas labākai tālāko nodaļu izpratnei.

1.1 Metamodelēšana, metamodeļi un modeļi

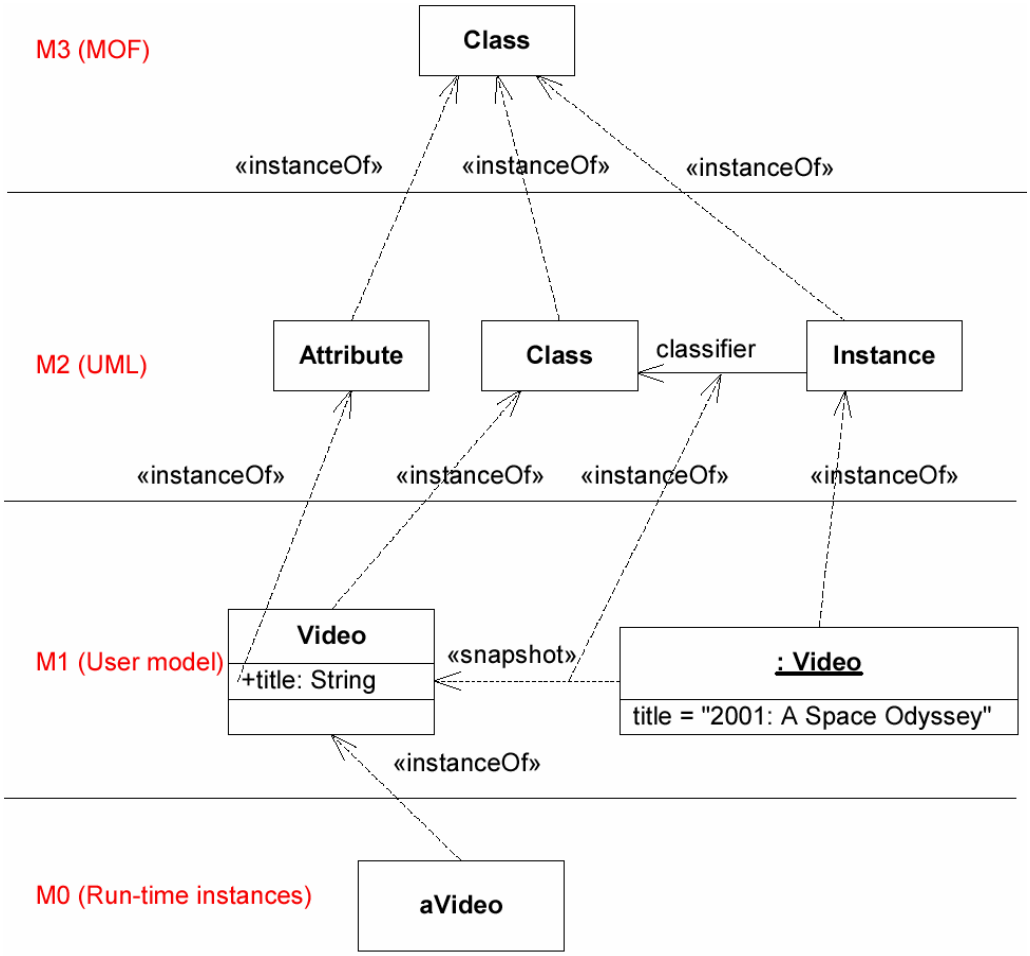
Ar modeli datorsistēmu būvē parasti saprot teorētisku konstrukciju, kas attēlo kaut kādu konkrētu pasaules fragmentu. Modelis ir savstarpēji saistītu objektu kopa, kas attēlo kādu jēdzienu. Fiziski modelis parasti ir grafisks attēls, kurā pēc **noteiktiem likumiem** izveidoti un izvietoti dažādi grafiskie elementi. Jāpiezīmē, ka modeļa grafiskā reprezentācija nav obligāta prasība modeļu veidošanā. Modelis var būt arī tekstuāls.

Kā augstāk tika minēts, modeļi tiek veidoti stingrā saskaņā ar noteiktiem modelēšanas likumiem. Kas nosaka šos likumus? Vispārīgi runājot, var teikt, ka katrs modelis tiek veidots saskaņā ar noteikumiem, kādus nosaka kāds cits augstāka līmeņa modelis – **metamodelis**. Par metamodeli var domāt kā par vispārīgu shēmu vai valodas sintaksi, saskaņā ar kuru tiek veidoti modeļi. Piemēram, viens metamodelis var tikt izmantots uzņēmuma organizatoriskās struktūras attēlošanā, cits metamodelis – automobiļu būves mašinērijā, vēl cits – informatīvās sistēmas aprakstā. Katrā no šiem gadījumiem metamodelis pārstāv vairākus iespējamos modeļus un nosaka likumus, pēc kuriem vadoties iespējams veidot jaunus modeļus, kas atbilstu dotajam metamodelim, vai citiem vārdiem sakot, būtu šī metamodeļa instances. Piemēram, izmantojot uzņēmuma organizatoriskās struktūras attēlojošu metamodeli, katrs uzņēmums var veidot savu modeli, vadoties pēc šī viena kopējā metamodeļa. Dažādiem uzņēmumiem šie modeļi iznāks dažādi, bet tos vienos kopējs metamodelis un kopēji zināmi likumi. Plašākā nozīmē **modelis** var būt jebkāda veida **formalizēts** (datoram saprotams) artefakts, kas parādās izstrādes gaitā un kura struktūru var aprakstīt ar metamodeli.

Metamodelis tiek saukts arī par modeļu modeli. Respektīvi, metamodelis pats ir modelis, kurš ir izveidots saskaņā ar kādu augstāka līmeņa metamodeli (metametamodeli). Katrs konkrēts metamodelis un tā atbilstošie modeļi definē divus metamodelēšanas abstrakcijas līmeņus. Teorētiski mēs varam runāt par patvaļīgi daudz metamodelēšanas līmeņiem, kur katrs nākamais ir iepriekšējā līmeņa meta līmenis. Praksē gan parasti tiek lietoti tikai četri abstrakcijas līmeņi: M0 – modeļu

instanču līmenis, M1 – modeļu līmenis, M2 – metamodeļu līmenis un M3 – metametamodeļu līmenis.

Patlaban visplašāk lietotais metamodelēšanas standarts ir OMG[19] izstrādātais MOF[23] (*Meta-Object Facility*) standarts, kurš nosaka četrus abstrakcijas līmeņus (skatīt 1.1. attēlu). Aktuālā MOF versija patlaban ir 2.0 [24], drīzumā tiek plānots pabeigt darbu pie versijas 2.1, bet ir sagaidāms, ka tas īpaši nemainīs MOF lietojamību praksē.



1.1. attēls. OMG metalīmeņu hierarhijas piemērs (attēls no [27])

Praksē daudzi modeļi ir aprakstīti kādā no modelēšanas valodas **UML**[25, 26, 27, 28, 29] versijām (t.i., atbilst UML metamodelim, kurš savukārt ir definēts MOF). Jāpiezīmē, ka modeļi var būt aprakstīti arī citās valodās, tai skaitā kādā no objektorientētām (OO) programēšanas valodām (konkrētas OO programmēšanas abstraktā sintakse arī atbilst metamodelim).

Daži tipiski modeļu piemēri:

- UML aktivitāšu diagramma – biznesa procesa modelis.
- UML lietojumu un aktivitāšu diagrammu kopums – sistēmas prasību specifikācija.
- UML klašu diagrammu kopums – sistēmas analīzes modelis.
- UML klašu diagrammas, kur lietoti J2EE stereotipi, – projektēšanas modelis.

- Java programma (pierakstīta abstraktā sintaksē).
- J2EE programma (komponentes + deskriptori), arī abstraktā sintaksē.
- Darba plūsmas (*workflow*) definīcija kādā no to definēšanas valodām (piemēram, BPMN valodā [30]).
- Tīmekļa lappuses formālā shēma (bez "dekorācijām").

1.2 Modeļu balstīta sistēmu izstrāde

Deviņdesmito gadu beigās un šā gadu desmita sākumā bija izveidojusies situācija, kad objektīvi bija uzkrāta nepieciešamā pieredze, lai ieviestu jaunu paradigmu un jaunus principus programmatūras izstrādē. Bija labi apgūtas tādas tehnoloģijas un zināšanas kā metamodelēšana, darbs ar modeļiem, UML diagrammu lietojums programmatūras izstrādē, dažādas komponenšu izstrādes vides (EJB, .NET, CORBA), objektorientētās programmēšanas valodas, tomēr tas viss kopā nebija devis sagaidāmo efektu programmatūras izstrādē, nebija panākts būtisks efektivitātes lēciens.

2000. gadā OMG sāka jaunu lielu projektu – Modeļbalstīta Arhitektūra (MDA[31, 32] – *Model Driven Architecture*). Kaut kas konkrēts gan parādījās tikai 2001. gada otrajā pusē, kad OMG publicēja MDA rokasgrāmatas pirmo versiju [33], kurā tika izklāstītas MDA pamatidejas un pielietojumi. Galvenā bija ideja, ka netriviālu sistēmu izstrādē vajadzētu sistemātiski izmantot dažādus meta-modelēšanas primitīvus. Svarīga bija atziņa, ka praksē modeļi netiek pienācīgi lietoti, netiek izmantots to potenciāls un ka modeļiem sistēmu izstrādes gaitā ir dažāda loma. Kopš tā brīža MDA un ar to saistītās lietas ir strauji attīstījušās, un arī vēl šodien to attīstības tempi nav mazinājušies.

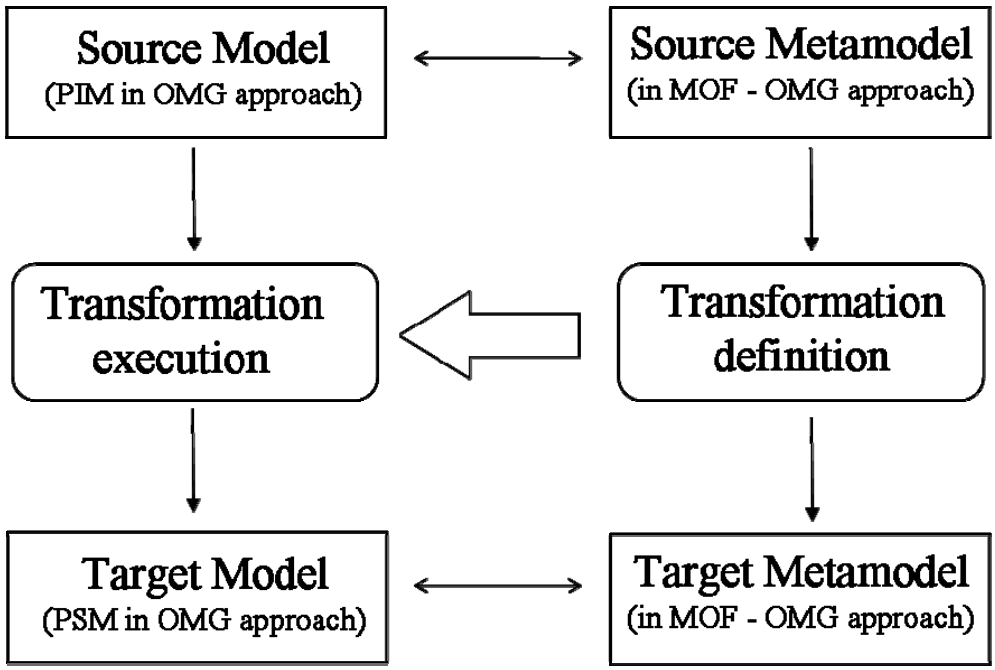
Interesanti, ka intuitīvi līdzīgas idejas daudzi jau pielietoja ikdienas praksē, bet nebija neviena veiksmīga mēģinājuma to formalizēt, sakārtot, izstrādāt metodiku, kā to visu konsekventi lietot programmatūras sistēmu izstrādē.

MDA ieviesa modeļu lomu jēdzienu sistēmu izstrādes procesā. Tika piedāvāti trīs veidu modeļi:

- Platformas neatkarīgais modelis – **PIM** (*Platform Independent Model)* , tas ir modelis, kurā neparādās platformas specifiskas lietas.
- Platformas atkarīgais modelis – **PSM** (*Platform Specific Model)*, tas ir modelis, kurā tiek apvienotas PIM aprakstītās lietas ar konkrētu platformu (piemēram, EJB, .NET, *WebServices*, CORBA) specifiskiem jēdzieniem.
- Konceptuālais modelis – **CIM** (*Computation Independent Model),* tas ir sistēmas konceptuālais vai biznesmodelis. Jāatzīmē, ka, ja CIM modelis arī tiek lietots sistēmu izstrādes procesā, tad vairāk kā dokumentācijas sastāvdaļa un nevis kā formāls „datoram saprotams" dokuments.

Vēl viens ļoti svarīgs formālisms, kurš tika ieviests MDA pieejā, ir modeļu transformācijas. Modeļu transformācija ir process, kam ir ļoti būtiska loma modeļu balstītu sistēmu izstrādē. Izstrādes gaitā vienu modeli vajag pārveidot par otru atbilstoši kādai dotajai metodei, pārnesot un papildinot visu nepieciešamo informāciju uz otru modeli. Pats transformācijas process nozīmē kāda modeļa (ieejas modeļa) vai

metamodeļa (ieejas metamodeļa) pārveidošanu par kādu citu modeli (rezultāta modeli) vai metamodeli(rezultāta metamodeli). Izplatītākais gadījums, ar ko visbiežāk darbojas izstrādātāji, ir tieši modeļu transformācijas. Tas nozīmē – paņemt kāda fiksēta metamodeļa (ieejas metamodeļa) modeli un pārveidot to par kāda cita (vai arī tā paša) fiksēta metamodeļa (rezultāta metamodeļa) modeli (skatīt 1.2. attēlu). Jāatzīmē, ka arī pašu metamodeļu transformēšana nav nekas neparasts un ka tā ir praksē dažkārt lietota iespēja.



1.2. attēls. Transformāciju pielietošanas shēma.

Laika gaitā ir attīstījušās vairākas pieejas, kā sistēmu izstrādē tiek pielietoti modeļi un atbilstošās transformācijas. Klasiskā OMG izpratnē MDA aplikācija tiek projektēta tā, ka tā sastāv no viena platformas neatkarīga modeļa un viena vai vairākiem platformas atkarīgiem modeļiem, un pilnām to realizācijām katrā no platformām, kuras aplikāciju izstrādātājs ir izvēlējies atbalstīt. Turklāt pašā aplikācijas izstrādes procesā tiek pielietotas transformācijas, kuras ir izstrādes procesa sastāvdaļa. Jāpiezīmē, ka MDA pieejā tiek uzskatīts, ka vienīgie atļautie metamodelēšanas līdzekļi ir OMG piedāvātie (MOF, UML).

Savukārt modeļu balstītā sistēmu izstrādē (MDSD) uz to visu raugās no plašāka skatu punkta. MDSD pieejā nav „ciešas" piesaistes pie MOF un UML, bet gan var lietot patvaļīgus (protams formalizētus) metamodelēšanas līdzekļus. Jāpiezīmē, ka specifiskos problēmu apgabalos ir izplatīta prakse izmantot specializētas grafiskas modelēšanas valodas – domēnu specifiskas valodas[16] (DSL – *Domain Specific Language*), kuras parasti ir ļoti atšķirīgas no UML.

Otra būtiska atšķirība MDSD pieejā, salīdzinot to ar MDA, ir saistīta ar modeļu izmantošanu. MDSD pieejā ieejas un rezultāta modeļi ir jebkuri divi secīgi modeļi, kuri tiek lietoti izstrādes gaitā. Tas, kas ir PIM un kas ir PSM, patiesībā ir relatīvs jēdziens, kurš ir atkarīgs tikai no tā konkrētā abstrakcijas līmeņa, no kura mēs uz to skatāmies. Piemēram, pat tik specifisks modelis kā Javas programma, pierakstīta tās

abstraktā sintaksē, var būt gan kā PSM modelis, gan arī kā PIM modelis. Šāda Javas programma varētu būt kā PSM modelis projektējuma klašu diagrammai un kā PIM modelis kādai konkrētai izstrādes vides platformai, kura savukārt ir PSM modelis augstāk minētajai Javas programmai. Te iet runa tikai par loģisko modeļu ķēdīti izstrādes gaitā. Jāpiezīmē, ka MDA ir tikai viens no MDSD pielietojuma variantiem.

Nodaļas beigās gribētu uzsvērt modeļu balstītu sistēmu izstrādes pieejas raksturīgākās īpašības:

- Metamodeļi un tiem atbilstošie modeļi ir galvenie programmatūras izstrādes artefakti. Tā ir jauna paradigma programmatūras izstrādē, jauns veids kā izstrādāt sistēmas. Šādi izstrādājot datorsistēmas, notiek "pacelšanās" citā abstrakcijas līmenī (tiek strādāts ar metamodeļiem un modeļiem).
- Transformāciju izmantošana. Pielietojot MDSD pieeju, ir nepieciešams formāli definēt transformācijas, turklāt transformācijas nav modeļa, bet gan izstrādes procesa daļa.

1.3 Modeļu transformāciju valodas

Kā jau tika minēts iepriekšējā nodaļā, lietojot modeļu balstītu izstrādes pieeju, vienu modeli vajag transformēt (pārveidot) par otru atbilstoši kādai dotajai metodei, pārnesot un papildinot visu nepieciešamo informāciju uz otru modeli. Jāpiebilst, ka šādas modeļu transformācijas varētu būt gan manuālas (līdz MDSD lietotais gadījums), gan arī automātiskas, ar formālu transformāciju palīdzību veiktas transformācijas (MDSD pieejā lietotais gadījums). Savukārt, lai varētu kaut kā definēt transformācijas procesus, ir nepieciešama modeļu transformāciju valoda. Modeļu transformāciju valoda ir programmēšanas valoda, kas paredzēta modeļus transformējošu procesu aprakstīšanai. Programma, kas rakstīta kādā no modeļu transformāciju valodām, tipiskākajā gadījumā kā ieejas datus saņem kāda metmodeļa modeli, bet kā izejas datus atgriež kāda cita (vai tā paša) metamodeļa modeli (skatīt 1.2. attēlu).

Tātad, lai varētu realizēt MDSD idejas praksē, bija nepieciešama speciāla veida, tieši šiem mērķiem paredzēta programmēšana valoda – modeļu transformācija valoda. Praksē izrādījās, ka neviena no esošajām programmēšanas vai modelēšanas valodām nebija pietiekami piemērota tām prasībām, kuras bija nepieciešamas modeļu transformācijās, un OMG 2002. gada aprīlī izsludināja konkursu uz šādas valodas standartu – QVT (*Queries/Views/Transformations* [34]).

Modeļu transformāciju valodas ir principiāli jauna valodu klase. Galvenās prasības, kurām jāatbilst modeļu transformāciju valodām ir sekojošas:

- Tām jāapkalpo patvaļīgi metamodeļu modeļi (OMG izpratnē gan tikai MOF metamodeļiem atbilstošie).
- Tām jāapstrādā ieejas modeļi un jāproducē rezultāta modeļi (jāvar aprakstīt un izpildīt transformācijas starp modeļiem, kuri atbilst noteiktiem metamodeļiem). Valodas konstrukcijām faktiski jābūt spējīgām strādāt ar metamodeļa (klašu) instanču kopām, jāprot tās atpazīt un pārveidot.

- Tām jābūt metamodeļbāzētām, respektīvi, arī pati transformāciju programma, kas rakstīta kādā no transformāciju valodām, ir uztverama kā modelis, kas atbilst dotās transformāciju valodas metamodelim.
- Tās varētu būt gan grafiskas, gan tekstuālas.
- Tām jābūt "viegli" saprotamām gan cilvēkiem, gan datoriem (jābūt pēc iespējas deklaratīvām).
- Jābūt atbilstošu rīku atbalstam, lai varētu šīs transformācijas ērti izveidot, mainīt un izpildīt.

Tika iesniegti vairāki projekti par valodas standartu, kuri laika gaitā vai nu tālāk neattīstījās, vai nu apvienojās, līdz palika viens standarta projekts – MOF QVT[35], kura izstrādē piedalās 16 iestādes, tostarp arī IBM, Sun un četras universitātes. Šis projekts apvieno daudzus sākotnējos projektus. 2005. gada martā standarta valodai pēc plāna bija jābūt gatavai, bet tā šobrīd (2007.g.) tā vēl joprojām ir tikai projekta stadijā un ir nodota OMG platformas uzdevumu komitejai (*ptc – Platform Task Committee*) galīgā varianta izstrādei [35].

Vienlaicīgi tika izstrādātas arī virkne citu ar OMG konkursu tieši nesaistītu modeļu transformāciju valodas – MOLA[6-13], Lx[36], GReAT[37, 38], UMLX[39], ATL[40, 41], TRL[42], Tefkat[43], AGG[44], MTF[45], ATOM[46], VMTS[47], BOTL[48], YATL[49], Fujaba[50], VIATRA2[51], RubyTL[52], ALAN[53], MT[54] un citas. Interesanti, ka vēl joprojām tiek izstrādātas arvien jaunas un papildinātas jau esošās modeļu transformāciju valodas. Starp minētajām valodām ir gan grafiskas, gan tekstuālas valodas. Īpaši ir jāpiemin grafiskā modeļu transformāciju valoda MOLA, kuras izstrādāšana ir daļa no autora veikto pētījumu rezultātiem promocijas darba ietvaros. Šajā jomā veikto pētījumu un valodas MOLA konspektīvs apraksts ir dots kopsavilkuma 3. nodaļā.

2 Redaktoru definēšanas valoda un universālais modelēšanas rīks

Šajā nodaļā ir apskatīti promocijas darba pirmā pētījuma posma (2000.g. – 2004.g.) rezultāti, kas ir apkopoti piecās publikācijās [1,2,3,4,5] un referēti četrās starptautiskās konferencēs. Autors ir aktīvi piedalījies visos pētījumos šajā laika posmā, sākot ar 2001. gadu.

2.1 Redaktoru definēšanas valoda un universālais modelēšanas rīka prototips.

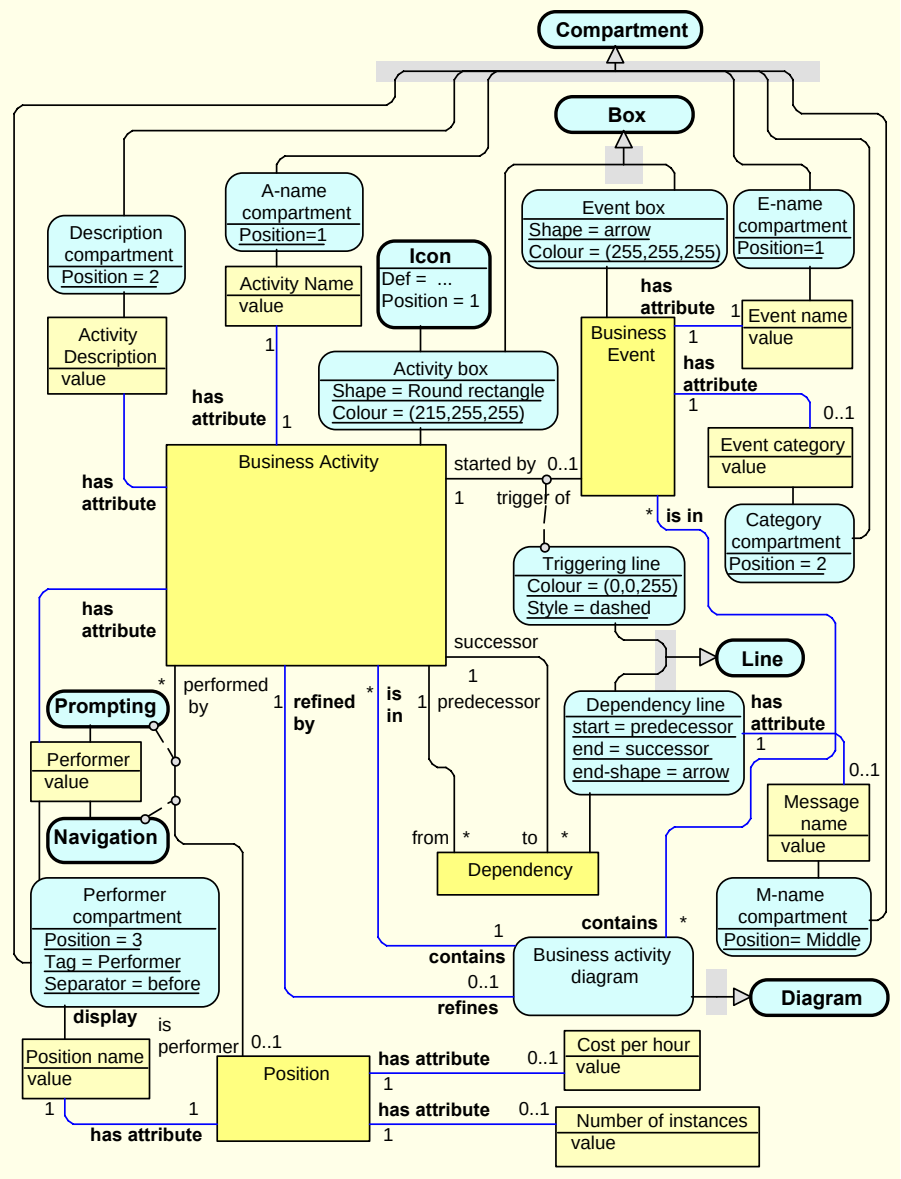
Šajā laika posmā tika aizsākti pētījumi, kuri noveda pie modeļu transformāciju valodas MOLA izstrādāšanas. Sākot pētījumus, apskatāmais problēmu apgabals bija diagrammu apstrādes rīki. Praksē bija nepieciešamība pēc elastīga (*flexible*) diagrammatiska skatu attēlotāja (redaktora), kurš būtu balstīts uz kādu konkrētu metamodeli [57]. Pētījumu rezultātā tika izstrādāta metodika un līdzekļi šādu diagrammatisku redaktoru definēšanai, tika izstrādāts universālais metamodeļbāzētais modelēšanas **rīka prototips** (diagrammatiskais skatu attēlotājs). Zemāk ir minēti galvenie izstrādāto līdzekļu elementi:

- **Loģiskais metamodelis** – diagrammu objektu loģiskās struktūras definēšana, izmantojot UML klašu diagrammas. Loģisko metamodeli mūsu patreizējā terminoloģijā mēs sauktu par domēna metamodeli. Tas apraksta kaut kāda noteikta problēmu apgabala jēdzienus un attiecības starp tiem.
- **Redaktoru definēšanas valoda** (EdDL – *Editor Definition Language*) – valoda, ar kuras palīdzību definē objektu grafisko reprezentāciju un redaktoru uzvedību (dinamiku). Ar šīs valodas palīdzību var aprakstīt šādas redaktora sastāvdaļas un īpašības – loģisko elementu grafisko reprezentāciju, elementu sastāvdaļu reprezentāciju, diagrammu elementu detalizēšanu, izmantojot apakšdiagrammas, dažādus priekšāteicējus (*prompting*), navigāciju starp diagrammām, simbolu paleti un dažas citas mazāk svarīgas īpašības.
- **Anotāciju kompilētājs** (*EdDL parser*).
- **Redaktoru dzinējs** – programmatūra, kas nodrošina redaktoru darbināšanu.
- **Grafiskais diagrammu dzinējs** (GDE – *Graphical Diagramming Engine*) – programmatūra, kas realizē diagrammu zīmēšanu, automātisko izvietošanu un citu ar grafisko objektu manipulācijām saistītu funkcionalitāti.

Lielākais autora ieguldījums šajā pētījumu posmā ir EdDL pamatkonstrukciju izstrāde un paša autora izprojektētie un realizētie **anotāciju kompilētājs** un **redaktoru dzinējs**.

Lai uzbūvētu redaktoru, bija nepieciešams veikt vairākus savā starpā saistītus rīka definēšanas posmus (skatīt arī 2.2. attēlu):

- Sākumā tika izveidots metamodelis (vai tā skats vienai diagrammai) kā UML klašu diagramma.

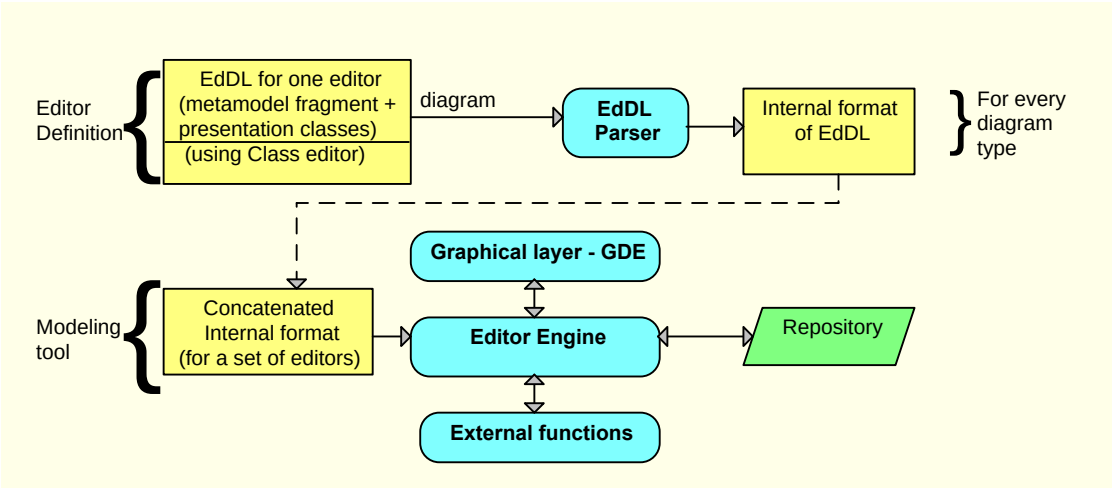


2.1. attēls. Ar EdDL anotēts loģiskais metamodelis.

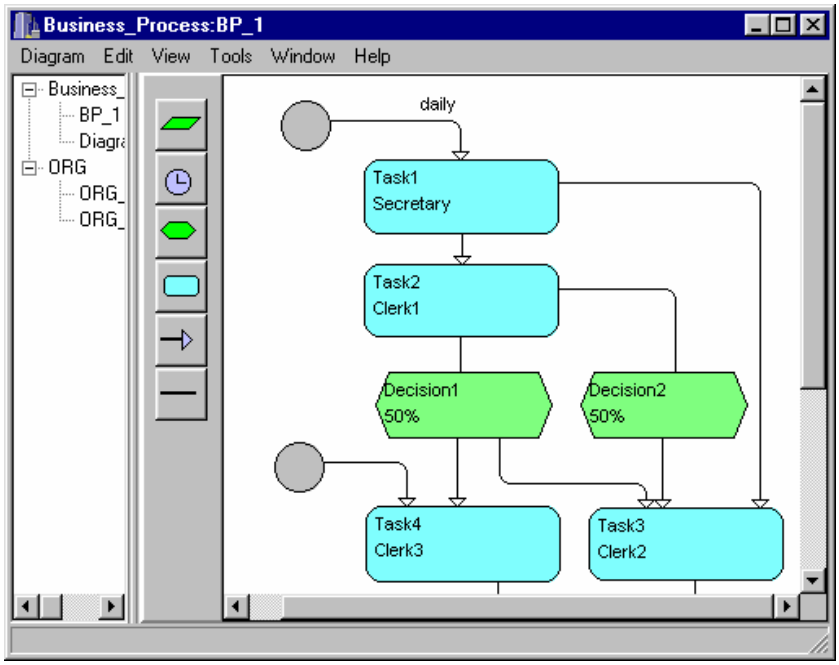
- Pēc tam, izmantojot EdDL konstrukcijas, metamodelis tika papildināts ar nepieciešamo grafiskās reprezentācijas specifikāciju. Respektīvi, metamodelis tika anotēts (skatīt 2.1. attēlu). Attēlā ir redzams, kādi ir daži no piedāvātajiem EdDL valodas līdzekļiem. Pamatideja bija ieviest metamodelī prezentācijas līmeņa elementus, piemēram, tādus kā `Box`, `Line`, `Compartment`, `Diagram`, u.c., kuru atbilstošajās apakšklasēs tika glabāta visa konkrētā būvējamā diagrammatiskā rīka grafisko elementu prezentācijas informācija. Savukārt, prezentācijas elementi pēc tam ar EdDL valodas līdzekļiem tika „sasaistīti" ar atbilstošajām domēna klasēm, izmantojot attēlojuma (*mapping*) asociācijas. Piemēram, klasei `Business Activity` ir pateikts, ka tās instances ir jāattēlo kā kastes (piesaistītā klases `Box` apakšklase `Activity Box`) ar atbilstošu noklusēto izmēru un krāsu. Vēl mēs redzam, ka šai klasei ir virkne atribūtu (prezentācijas līmeņa terminos – *compartments*), kuri arī ir atbilstoši anotēti. Šādai atribūtu anotēšanai sekas bija tādas, ka bija izdevīgi veidot nedaudz „dīvainu" klašu diagrammu, kurā domēnu klašu atribūti netiek

attēloti pašā klasē. Jāpiezīmē, ka šī un virkne citu neērtību tika novērstas nākamo pētījumu rezultātā, kad prezentācijas un domēna metamodeļi tika nodalīti viens no otra.

- Nākamais solis, būvējot specifisku diagrammatisku redaktoru, ir šādi anotētu metamodeļu kompilācija. To realizē anotāciju kompilētājs (*EdDL parser*), kurš anotēto metamodeli pārvērš tā iekšējā formātā, kuru „saprot" redaktoru dzinējs. Izpildlaika komponente – redaktoru dzinējs interpretē šo iekšējo formātu, un gala lietotājs "saņem" gatavu diagrammatisku rīku (skatīt 2.3. attēlu).



2.2. attēls. Diagrammatiskā attēlotāja galvenie elementi.



2.3. attēls. Noģenerētais redaktors darbībā.

Jāpiezīmē, ka bija jau izstrādāta virkne rīku, kuri risināja līdzīgas problēmas. Galvenā to atšķirība no MII izstrādātā universālā modelēšanas rīka prototipa bija tā, ka tie tipiski bija konstruēti kā gatavi redaktoru komplekti ar plašām to konfigurēšanas iespējām. Tos varēja pielietot tikai iepriekš definētam problēmu apgabalam. Turpretī, MII izstrādātais rīks bija daudz universālāks. Tas balstījās uz loģisko metamodeli un relatīvi neatkarīgu redaktoru definēšanas valodu EdDL, kas deva iespēju rīku izmantot plašam grafisku redaktoru lokam. Daži interesantākie tā laika rīki, kuri pretendēja uz līdzīgu problēmu apgabalu bija:

- **Metaedit**, izstrādātājs MetaCase Consulting [58]
- **KOGGE**, izstrādātājs University of Koblenz [59]
- **Toolbuilder**, izstrādātājs Lincoln Software [60]
- **DOME**, izstrādātājs Honeywell [61]
- **Moses**, izstrādātājs ETH Zurich [62]

2.2 Universālais modelēšanas rīks

Pēc pirmajiem iegūtajiem pētījumu rezultātiem par EdDL un tās pielietošanu diagrammatisku redaktoru būvē radās pārliecība par iespējām un idejas, kā šo valodu varētu paplašināt un pilnveidot, lai to varētu izmantot plašākā kontekstā. Problēmu izpētes apgabals bija universālu modelēšanas rīku būve. Detalizētāk tika pievērsta uzmanība tieši biznesa procesu modelēšanas rīku jomai, kur MII bija jau uzkrāta liela pieredze.

Situācija, kāda bija izveidojusies modelēšanas rīku jomā uz to brīdi, bija tāda, ka neviens no pieejamiem rīkiem neapmierināja visas prasības tai konkrētā apgabalā, kuram tas bija paredzēts. Katram no tiem bija savas stiprās un vājās vietas. Situācija īpaši sarežģīta bija domēnu specifisku valodu [16] (*DSL – domain specific language*) lietošanā uzņēmumos. Katra no šādām valodām bija ar kaut ko unikāla, atšķirīga no citām. Pat tādā nosacīti stabilā apgabalā kā UML bija problēmas ar rīku lietošanu. Tipiski, ka lielos uzņēmumos UML rīkiem bija nepieciešami dažādi specifiski papildinājumi, kuri nebija realizējami, izmantojot esošo rīku piedāvātos paplašinājuma servisus.

Ir iespējami dažādi veidi, kā risināt augstāk minētās problēmas modelēšanu rīku jomā:

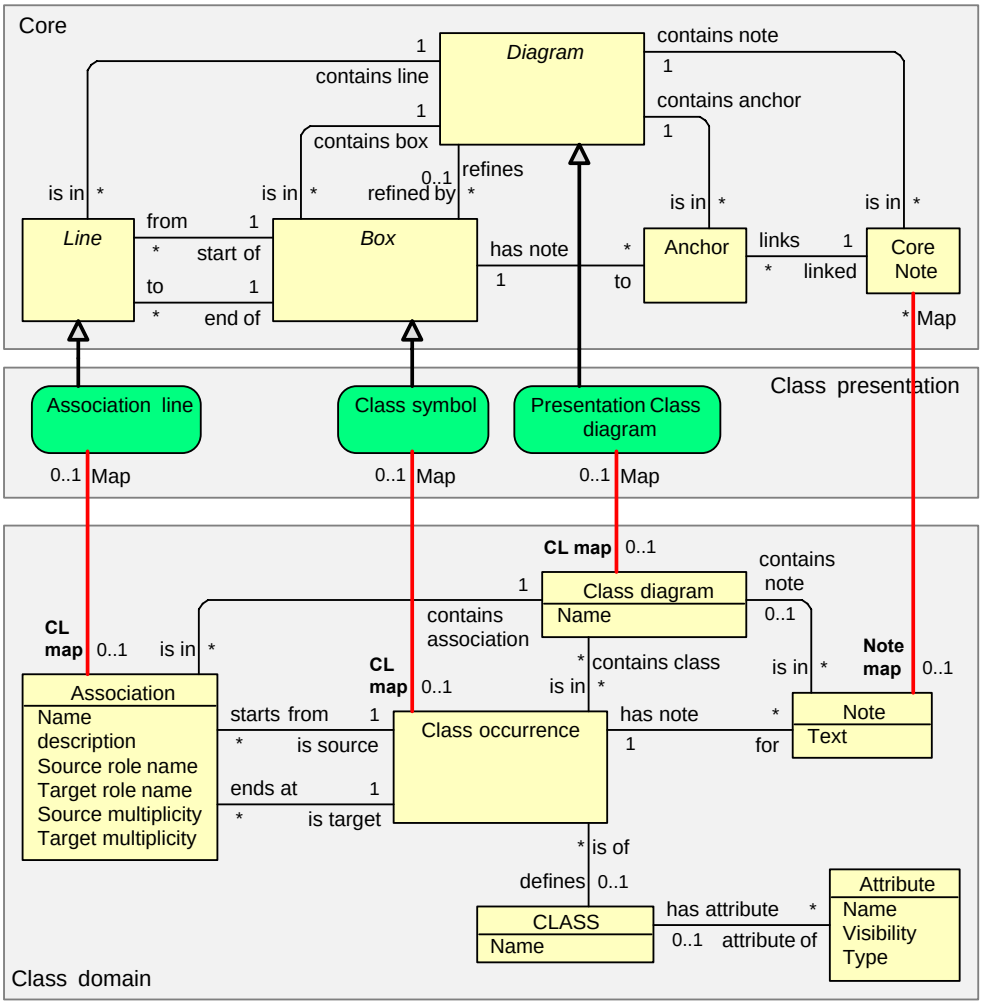
- Var izstrādāt rīku individuāli katrai modelēšanas metodei.
- Var mēģināt izstrādāt maksimāli universālu modelēšanas rīku, kurš apmierinātu pēc iespējas plašāku modelēšanas metožu spektru.
- Var mēģināt izstrādāt pilnībā metamodeļbāzētu universālu rīku būves platformu, kurā nebūtu iebūvētas nevienas iepriekš definētas modelēšanas metodes. Savukārt, izmantojot šādu platformu, varētu ātri un lēti izveidot nepieciešamos rīku komplektus.

Pētot šo problemātiku un balstoties uz pieredzi, kura tika iegūta, izstrādājot universālā modelēšanas rīka prototipu, tika secināts, ka saprātīgākais un arī interesantākais problēmas risinājums būtu izstrādāt universālu rīku būves platformu. Tika apzinātas prasības, kuras šādai rīku būves platformai būtu jāapmierina. Zemāk ir minētas galvenās no tām:

- Rīkam jābūt pilnībā metamodeļbāzētam. Tas nozīmē, ka rīkam nav predefinētas (iebūvētas) modelēšanas metodoloģijas. Lai sāktu kaut ko modelēt rīks sākumā ir "jāuzpilda" ar metamodeli un papildus informāciju, kura apraksta vēlamo modelēšanas metodoloģiju.
- Rīkam jāatbalsta dažādas modelēšanas notācijas, izmantojot kopēju domēna metamodeli. Tas nozīmē, ka lietotājs var izveidot modeli vienā notācijā un strādāt ar to pēc tam citā notācijā (ideja tā arī netika līdz galam realizēta – nebija vēl izprasta transformāciju nozīme un to lietojums).
- Rīkam jānodrošina universālas metamodeļbāzētas rediģēšanas iespējas. Tas nozīmē, ka rīkam, vadoties tikai no metamodelī esošās informācijas, ir jāpiedāvā lietotājiem sakarīgas domēna datu rediģēšanas iespējas.

Pētījumu rezultātā tika izstrādāta metodika un līdzekļi (platforma) šādu universālu modelēšanas rīku būvei – universālais metamodeļbāzētais modelēšanas rīks (*Exigen Business Modeler*). Zemāk ir minētas galvenās izstrādātās platformas sastāvdaļas:

- **Domēna metamodelis** (*domain metamodel*). Tas definē modelēšanas jēdzienus (konceptus), to atribūtus (datus) un attiecības starp tiem (skatīt 2.3 attēla apakšējo daļu).

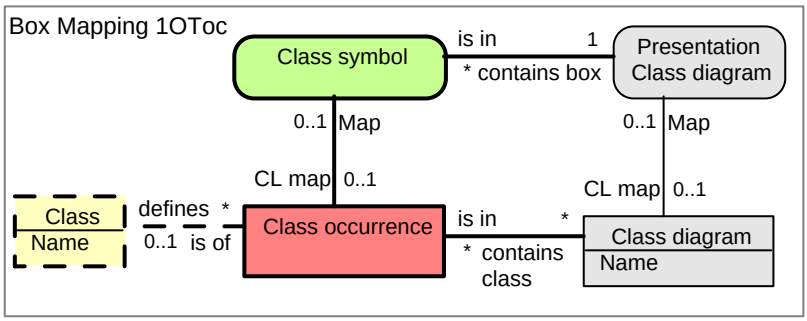


2.3 attēls. Paplašinātā metamodeļa piemēra fragments (vienkāršotai klašu diagrammai).

- **Paplašinātais metamodelis** (*extended metamodel*). Sastāv no divām būtiskām daļām – metamodeļa prezentācijas (*presentation*) daļas un metamodeļa kodola

(*core*) daļas (skatīt 2.3 attēlu). Paplašinātā metamodeļa klases tiek savienotas ar atbilstošajām domēna metamodeļa klasēm, izmantojot attēlojuma (*mapping*) asociācijas.

- **Diagrammu anotāciju definēšanas** (attēlojumu definēšana) un to izpildlaika interpretēšanas serviss. Serviss nodrošina iespēju aprakstīt, kā loģiskie elementi (modelēšanas jēdzieni) tiek attēloti par diagrammas elementiem (*mapping*), kā arī apraksta rīka uzvedību darbam ar diagrammām (skatīt 2.4 attēlu). Šai platformas sastāvdaļai bija īpaša loma vēlāko pētījumu kontekstā, kuri bija saistīti ar modeļu transformāciju valodu MOLA. Tā ir svarīgākā diagrammu definēšanas daļa universālā rīkā. Ar attēlojumu palīdzību tiek aprakstīts tas, kā loģiskie (domēna) elementi tiek attēloti par diagrammas elementiem. Attēlojumu sintakse (skatīt 2.4. attēlu) tiek definēta ar attēlojumu tipiem (paraugiem) (*mapping types (patterns)*). Attēlojuma tips satur references uz metamodeļa klasēm un asociācijām, kas ir kaut kas ļoti „tuvu stāvošs" parauga jēdzienam, kas vēlāk tika izmantots valodā MOLA. Savukārt, attēlojuma semantika ir atkarīga no atbilstošā attēlojuma tipa, un tā nosaka, kādas semantiskas darbības ir jāveic pie diagrammu izmaiņām. Mūsdienu izpratnē šīs darbības bija programmistiski nokodētas („iešūtas") transformācijas (*hard-coded transformations*). Savukārt, valodas MOLA kontekstā tās būtu darbības, kuras ir jāveic atbilstošajā likumā, kurā atrodas dotais valodas MOLA paraugs. Publikācijā [4] ir detalizēti aprakstīts, kā var nodefinēt attēlojuma formālo semantiku, lietojot OCL [63] valodu. Respektīvi, katram attēlojuma tipam var pierakstīt tam atbilstošo OCL izteiksmi. Rīks ierobežotā apjomā nodrošināja arī iespēju papildināt esošos attēlojuma tipus ar papildus OCL izteiksmēm.



2.4 attēls. Attēlojuma piemērs (1OToc attēlojums).

- **Diagrammu elementu grafiskā stila** definēšanas un interpretēšanas serviss.
- **Modeļu koka pārvaldības dzinējs** – elastīga metamodeļbāzēta modeļa koka definēšana un interpretēšana. Modeļa koka definēšanā tiek izmantota virkne iespēju, kuras kombinējot ir iespējams nodefinēt praktiski jebkādu sakarīgu modeļa pārskata koka struktūru. Biežāk lietotie līdzekļi, definējot modeļa koku ir:
  - Fiksētie koka elementi – izmanto, lai veidotu modeļa koka fiksētu pamatstruktūru.
  - Objektu elementi – koka interpretācijas (izpildes) laikā tiek "aizpildīti" ar atbilstošajām metamodeļa klašu instancēm.
  - References elementi – izmanto rekursīvu koka elementu definēšanai.
  - Atlases kritēriji objektu elementiem – viena no „jaudīgākajām" koka definēšanas īpašībām. Atļauj uzdot specifiskus, OCL līdzīgus atlases kritērijus, balstītus uz metamodeļa asociācijām un klašu atribūtu vērtībām.

- Uznirstošo izvēlņu definēšanas līdzekļi – nodrošina iespēju izveidot lietotāju definētas izvēlnes modeļa koka elementiem.

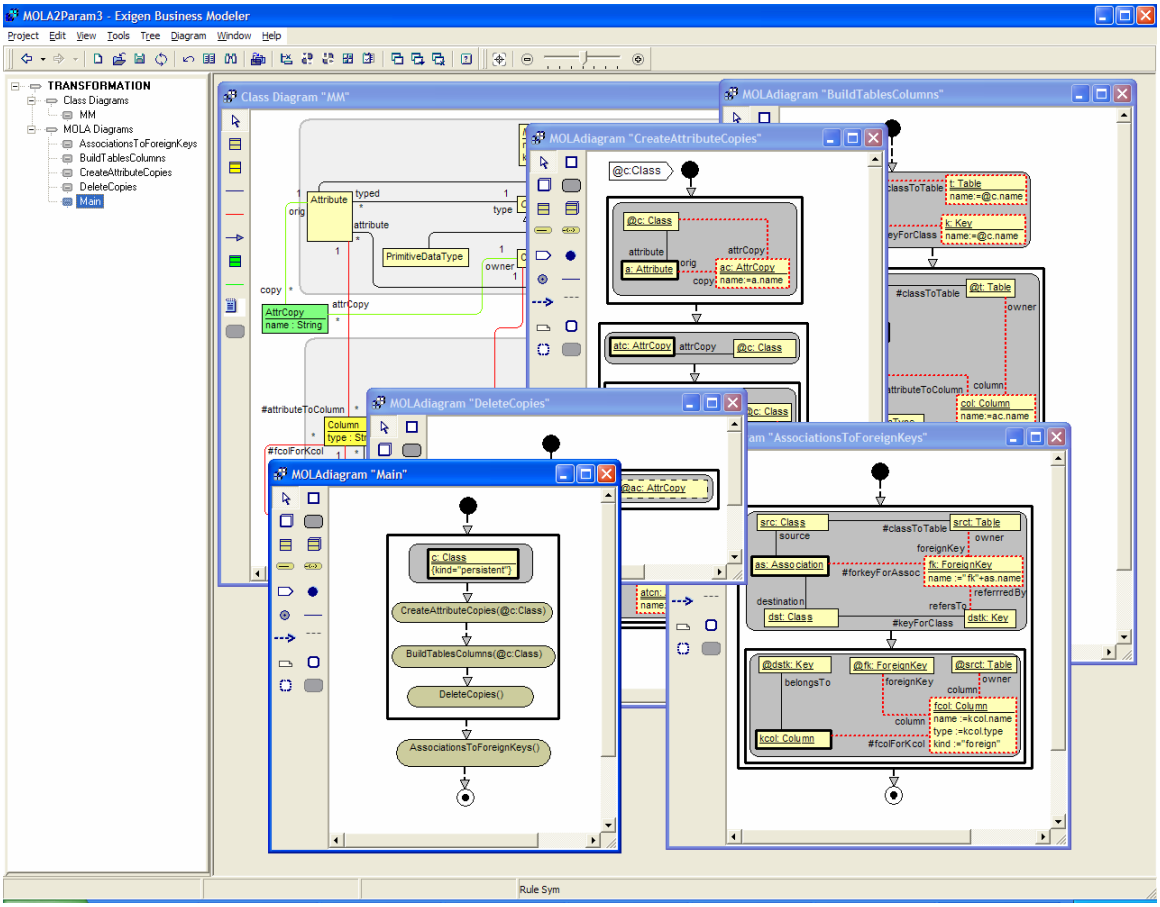
- **Universālo īpašību redaktoru dzinējs** (*generic property editor engine*) – nodrošina metamodeļbāzētu domēna datu redaktoru definēšanu un interpretēšanu. Universālais redaktors, izmantojot domēna metamodeļa klases informāciju (atribūtus un to tipus, asociācijas uz citām klasēm, kardinalitātes u.c.), noģenerē dialogu logus (redaktorus), ar kuru palīdzību ir iespējams veikt domēna datu rediģēšanu. Redaktoru ģenerēšanā tiek pielietotas dažādas heiristikas, kuras ir pierādījušas savu lietderību turpmāko gadu laikā. Piemēram, viena no tām – ar kompozīcijas tipa asociāciju piesaistītās klases instanču kopa tiek attēlota redaktorā kā iekļautie tabulārie redaktori. Protams, ka šādi nevarēja realizēt visus nepieciešamos nestandarta gadījumus. Tos pēc patreizējās izpratnes vajadzētu realizēt, izmantojot modeļu transformācijas, bet tajā laikā vēl nebija tādu līdzekļu. Tomēr bija iespēja konfigurēt noģenerētos universālos redaktorus un vairumā gadījumu šādi varēja iegūt rūpnieciskajiem modelēšanas rīkiem līdzvērtīgu īpašību redaktoru kvalitāti. Valodas MOLA redaktors ir tieši tādā veidā uzbūvēts – izmantojot konfigurētos un daļēji arī nekonfigurētos universālos metamodeļbāzētos īpašību redaktorus (skatīt 2.5. attēlu) .
- **Universālo tabulāro skatu dzinējs** (*generic table editor engine*) – nodrošina metamodeļbāzētu domēna datu tabulāru redaktoru definēšanu un interpretēšanu. Tas tika realizēts, pielietojot līdzīgus principus kā universālajiem īpašību redaktoriem. Tā galvenā vērtība ir piedāvātā iespēja pārskatīt kādas metamodeļa klases visas instances ar to atbilstošajiem atribūtiem. Laika gaitā šī iespēja izrādījās ļoti noderīga kā noklusētais instanču pārlūks.

Lielākais autora ieguldījums šajā pētījumu posmā ir universālā metamodeļbāzētā modelēšanas rīka arhitektūras pamatprincipu izstrāde, kā arī autora devums, izstrādājot **diagrammu anotāciju definēšanas** (attēlojuma) pamatprincipus, un paša autora izprojektētie un realizētie **universālā modelēšanas rīka dispečers**, **modeļu koka pārvaldības dzinējs**, **universālais īpašību redaktoru dzinējs** un **universālais tabulāro skatu dzinējs**.

Izstrādātais universālais metamodeļbāzētais rīks nebija vienīgais pieejamais šāda veida rīks. Funkcionāli vistuvākie līdzinieki bija Metaedit+[64], GME[65] un ATOM3[66]. Mūsu izstrādātais rīks bija universālāks, elastīgāks un ar plašākām dažādu modelēšanas tehnoloģiju definēšanas iespējām. Izstrādātajā rīkā ir iekļauta attēlojumu tipu bibliotēka, kas nodrošina domēna elementu attēlojumu atbilstošajās diagrammās. Lietojot rīku, ir pierādījies, ka izvēlētais attēlojumu tipu komplekts ir praktiski pietiekams, lai varētu izveidot ļoti plašu modelēšanas rīku saimi. Savukārt citos funkcionāli līdzīgos rīkos ir ļoti trūcīgas attēlojumu definēšanas iespējas. Tajos ir iekļauti tikai paši elementārākie attēlojumu līdzekļi, piemēram, viens konkrēts domēna elements attēlojas par vienu prezentācijas (diagrammas) elementu. Jāpiezīmē, ka dažos rīkos ir iespējama attēlojumu tipu papildināšana, programmējot kādā no tradicionālām programmēšanas valodām, piemēram, C++ valodā rīkā GME. Tomēr tas ir izdarāms, lietojot neērtus un darbietilpīgus zema līmeņa programmēšanas līdzekļus. Jāuzsver arī fakts, ka promocijas darba ietvaros izstrādātajā rīkā, salīdzinot to ar citiem līdzīgiem rīkiem, var visātrāk izveidot praktiski lietojamu grafisku redaktoru. Piemēram, praktiski lietojama UML klašu redaktora izveide neaizņem vairāk kā trīs darba dienas.

Noslēgumā īss kopsavilkums par šo pētījumu laika posmu:

- Tika uzkrāta pieredze un zināšanas par to, kā vajag, un, protams, arī par to kā nevajag būvēt šāda veida universālus rīkus.
- Tika uzbūvēts universāls modelēšanas rīks (*Exigen Business Modeler*), kurš vēl joprojām aktīvi tiek izmantots dažādiem eksperimentiem MII pētījumu projektos. Jāpiezīmē, ka modeļu transformāciju valodas MOLA redaktors arī tika realizēts, izmantojot šo rīku (skatīt 2.5. attēlu).
- Tika ieviesti attēlojuma (*mapping*), parauga (*pattern*) un transformāciju jēdzieni:
  - Attēlojumu definēšana – parauga jēdziens, kaut kas ļoti „tuvu stāvošs" parauga jēdzienam, kas vēlāk tika izmantots valodā MOLA.
  - Attēlojumu realizācija – "iešūtas" transformācijas (*hard-coded transformations*). Valodas MOLA kontekstā tās būtu darbības, kuras ir jāveic atbilstošajā likumā, kurā atrodas dotais paraugs.
- **Ļoti svarīgi** – iegūtie rezultāti rosināja sākt domāt MDA un MDSD ideju kontekstā. Tika iegūtas zināšanas un prasme, kā var izmantot modeļu transformācijas rīku būvē, kā arī rasta izpratne par to, kādai jābūt transformāciju valodai, lai tā būtu ērti lietojama, tika izprasta arī transformāciju nozīme un to vieta programmatūras izstrādē. Jāatzīmē, ka šajā laikā posmā iegūtie pētījumu rezultāti bija nozīmīgs ideju un iedvesmas avots tālākajiem pētījumiem par modeļu transformāciju valodu MOLA.



2.5. attēls. Universālā modelēšanas rīka darbības piemērs (valodas MOLA redaktors).

3 Modeļu transformāciju valoda MOLA un tās lietošanas metodoloģija

Šajā nodaļā ir apskatīti promocijas darba otrā pētījuma posma (2003.g. – 2006.g.) rezultāti, kas ir apkopoti piecās publikācijās [6,7,8,9,10] un referēti piecās starptautiskās konferencēs. Autors ir aktīvi piedalījies visās ar šiem pētījumiem saistītās aktivitātēs.

Pētījumi par modeļu transformāciju valodu MOLA bija loģisks turpinājums aktivitātēm, kuras bija saistītas ar universālu modelēšanas rīku izstrādi. Kā jau tika minēts, tad valodas MOLA redaktors tika uzbūvēts, izmantojot iepriekšējā pētījumu posmā izstrādāto universālo metamodeļbāzēto modelēšanas rīku (skatīt 2.5. attēlu), kurš, sākot jau ar pirmajām eksperimentālajām valodas MOLA versijām, ir ļoti labi sevi pierādījis.

3.1 Modeļu transformāciju valoda MOLA

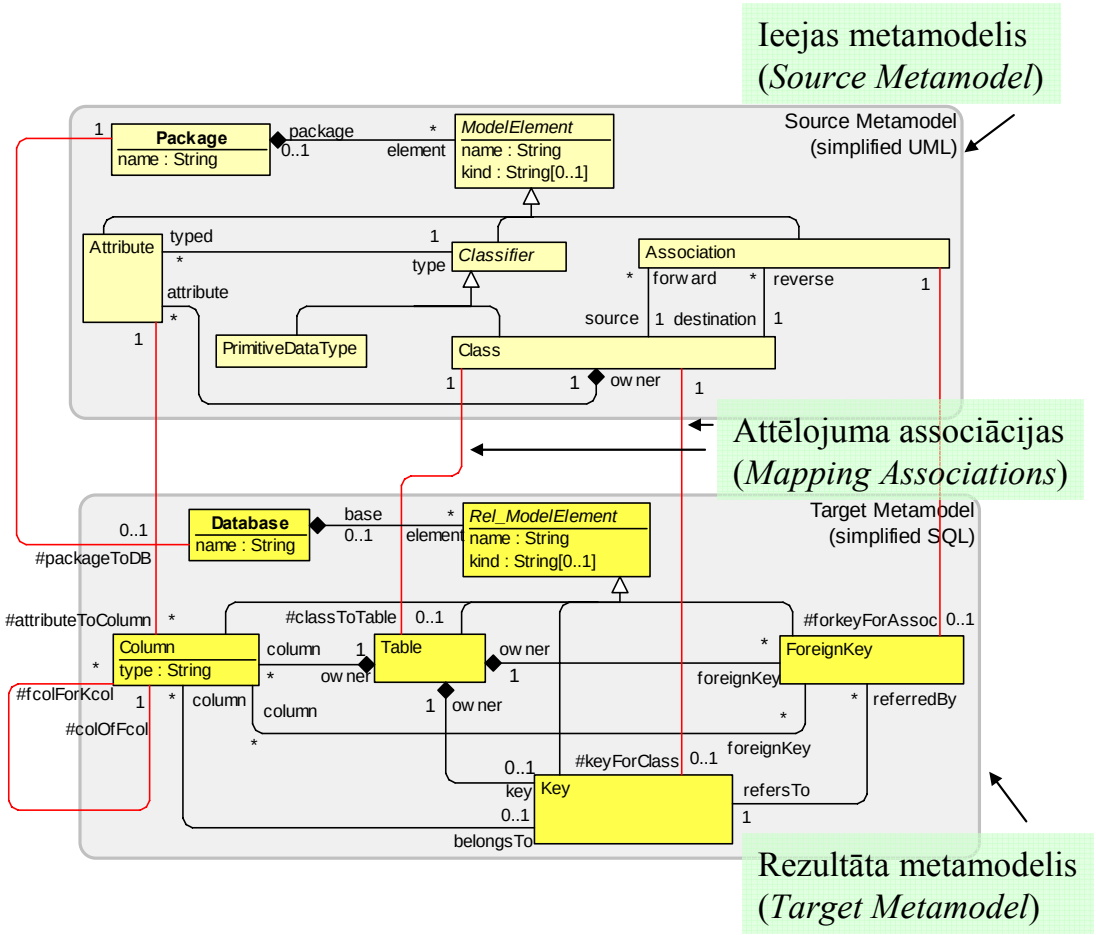
Iepriekšējo gadu pētījumu uzkrātā pieredze, iegūtie rezultāti un OMG MDA[33] iniciatīvā paustās idejas bija devušas izpratni un zināšanas par to, kā varētu pielietot modeļu transformācijas. Tika saprasta transformāciju nozīme un vieta programmatūras izstrādē. Bija rasta izpratne par to, kādai jābūt modeļu transformāciju valodai, lai tā būtu ērti lietojama modeļu balstītu sistēmu izstrādē (MDSD).

Projektējot promocijas darba ietvaros izstrādāto valodu MOLA, galvenie mērķi bija:

- Izveidot tādu valodu, kurā transformācijas būtu definējamas pēc iespējas ērtākā veidā.
- Transformācijām jābūt "viegli" saprotamām (lasāmām) gan cilvēkam, gan datoram.
- Valodai jābūt viegli implementējamai. Jāpiezīmē, ka tas vēlāk izrādījās ļoti svarīgs valodas aspekts.

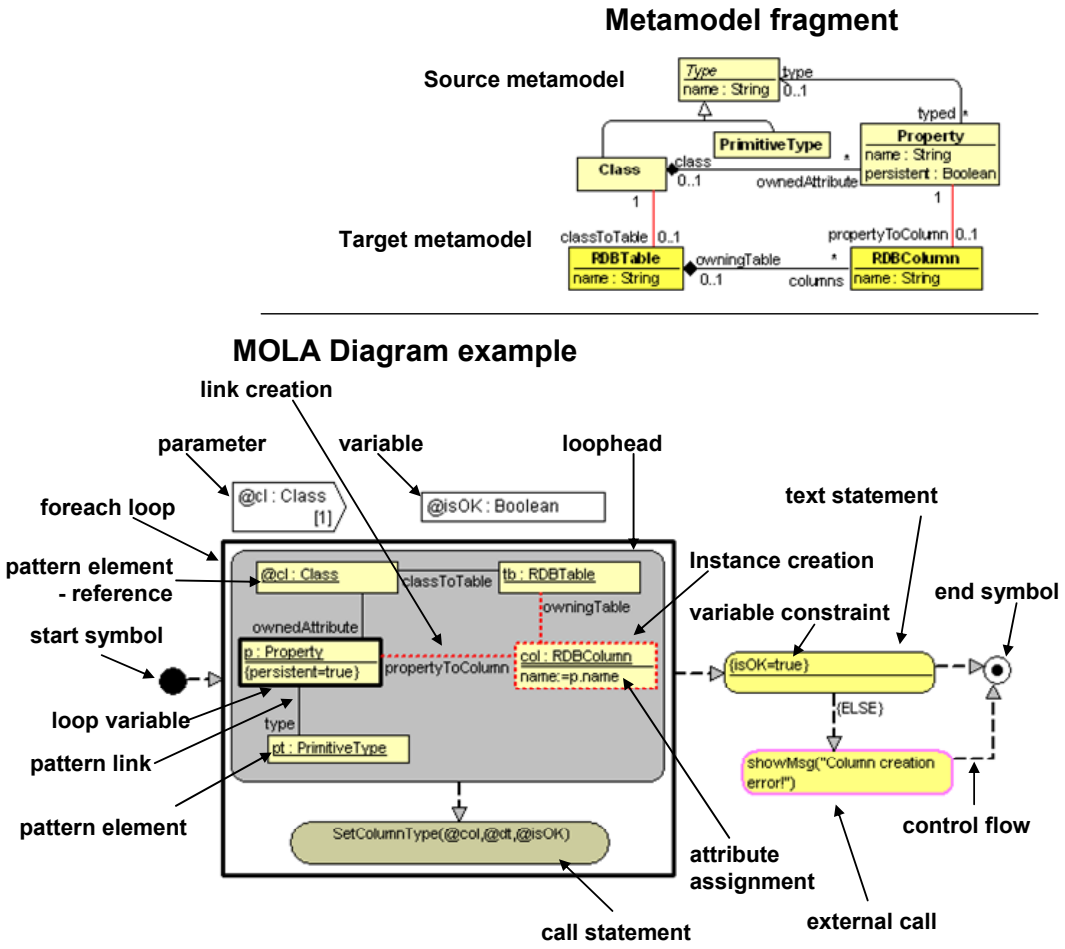
Izstrādātā valoda MOLA ir unikāla grafiska modeļu transformāciju valoda. Grafiski tiek attēloti tie valodas elementi, kas arī grafiski ir vislabāk saprotami. Valodā plaši tiek izmantota paraugu (*patterns*) lietošana, kura ir apvienota ar vienkāršām iteratīvām vadības struktūrām, kuras ir aizgūtas no tradicionālās strukturālās programmēšanas. Valodā ir ieviesti arī OO elementi, kuriem ir jēga dotajā kontekstā. Piemēram, tiek atbalstītas tādas klasiskas OO programmēšanas idejas kā mantošana un polimorfisms.

Katra MOLA programma definē vienu transformāciju un tā sastāv no metamodeļa un MOLA transformācijas. Ar metamodeli valodā MOLA tiek saprasta klašu diagramma, kurā ir ieejas metamodeļa un rezultāta metamodeļa daļas, un attēlojumu asociācijas (skatīt 3.1. attēlu). Vispārīgā gadījumā ieejas un rezultāta metamodeļa daļas var arī sakrist. MOLA transformācija ir viena vai vairākas MOLA procedūras (diagrammas), no kurām viena ir galvenā.



3.1. attēls. Ieejas un rezultāta metamodeļa piemērs.

Tālāk tiks dots īss pārskats par dažām biežāk lietotām valodas MOLA pamatkonstrukcijām (skatīt arī attēlu 3.2.). MOLA transformāciju programma ir grafisko instrukciju secība, kuras ir savienotas ar bultām. Viens no visbiežāk lietotajiem instrukciju tipiem ir `foreach` cikls. `Foreach` cikls satur cikla galvas instrukciju (*loop head statement*) un cikla mainīgo (*loop variable*). Cikli var būt arī iekļauti viens otrā. Cita ļoti svarīga valodas konstrukcija ir likumi (*rules*). Jāpiezīmē, ka cikla galva ir likuma specifisks gadījums. Likums ir paraugs (*pattern*), kas ir apvienots ar darbībām. Paraugiem valodā MOLA ir īpaša nozīme, tie nosaka instanču kopu, ar kurām tiks veiktas kādas darbības. Valodā MOLA paraugs ir metamodeļa fragments, kurā katram klases (parauga) elementam ir arī vārds un kurā katra klase var parādīties vairākas reizes ar kādu konkrētu lomu (līdzīgi kā UML komunikāciju diagrammās klasei pievieno lomas vārdu). Elementi ir pierakstīti instanču notācijā, un tie referencē atbilstošās metamodeļa klases. Savā starpā elementi var būt savienoti ar saitēm (*links*), kuras atbilst atbilstošajām metamodeļa asociācijām. Klases elementiem var būt arī nosacījumi (*constraints*), kuri tiek pierakstīti ļoti ierobežotā OCL apakškopā. Jāuzsver, ka cikli, likumi, paraugi un atbilstošās darbības, kuras ir jāveic ar instancēm, tiek attēlotas grafiski.



3.2. attēls. Valodas MOLA elementi.

Lai labāk saprastu valodas MOLA uzbūves pamatprincipus ir lietderīgi izklāstīt valodas izpildes vispārīgo semantiku:

- MOLA programma "saņem" ieejas modeli – instanču/saišu kopu, kura atbilst ieejas metamodelim.
- MOLA programma transformācijas rezultātā "izveido" rezultāta modeli – instanču/saišu kopu, kura atbilst rezultāta metamodelim.
- Transformācijas izpilde sākas ar galvenās programmas starta simbolu.
- Izsaukuma instrukcijas (*call statements*) izsauc pāreju uz atbilstošajām MOLA procedūrām (apakšprogrammām). Jāpiezīmē, ka izsaukumi var būt arī rekursīvi.
- Instrukciju izpilde ir secīga – cikls tiek izpildīts secīgi katrai "derīgai" cikla mainīgā instancei.

Tālāk tiek dots detalizētāks paskaidrojums par 3.2. attēlā redzamo MOLA transformācijas fragmentu. Attēlā ir redzams metamodeļa fragments un kāda valodas MOLA procedūra. Interesantākā procedūras daļa ir `foreach` cikls. Dotā cikla konstrukcija tiks izpildīta ar visām tām `Property` instancēm, kurām ir saite (saite atbilst asociācijai ar lomas vārdu `type` (skatīt metamodeļa fragmentu 3.2. attēla augšējā labajā stūrī)) ar kādu `PrimitiveType` instanci un saite ar tādu `Class` instanci `@cl`, kura ir saņemta kā parametrs, izsaucot šo procedūru. Savukārt, šai `Class` klases instancei (`@cl`) ir jābūt saistītai ar kādu `RDBTable` instanci. Vēl ir jāievēro, ka atbilstošai `Property` instancei ir jāapmierina nosacījums (*constraint*),

ka tās atribūta `persistent` vērtība ir `true`. Ja izpildās visi augstāk aprakstītie nosacījumi, tad tiek izpildīta likuma darbību daļa. Šai gadījumā tiek radīta klases `RDBColumn` instance (attēlota kā kaste ar sarkanu raustītu rāmīti) un šī jaunradītā instance tiek sasaistīta ar atbilstošajām `Property` un `RDBTable` klašu instancēm (raustītas sarkanas līnijas). Pēc tam tiek izsaukta cita MOLA procedūra – `SetColumnType` ar atbilstošajiem parametriem. Pēc `SetColumnType` izpildīšanas tiek meklēta nākamā derīgā `Property` klases instance.

Kā jau tika minēts 1.3. nodaļā, tad valoda MOLA nav vienīgā modeļu transformāciju valoda. Vairums no minētajām valodām (piemēram, MOF QVT[35], MOLA un daudzas citas) agrāk vai vēlāk kļuva par faktiski pilnām tradicionāliem modeļu transformāciju uzdevumiem (uzdevumus var adekvāti norealizēt ar programmām dotajā transformāciju valodā). Lielākā atšķirība starp valodām ir tā, cik viegli tajās var rakstīt un lasīt transformācijas. Praksē ir pierādījies, ka valodai MOLA, salīdzinot ar citām valodām, ir augsta izteiksmes spēja un lasāmība. Saprātīgi veidotas valodas MOLA transformācijas praktiski ir pašdokumentējošas, tajās ir viegli saprast, kādas darbības ar modeļiem tiek veiktas. Darbs ar studentiem ir pierādījis, ka valoda MOLA ir arī ātri iemācāma un viegli uztverama transformāciju valoda. Praktiski pietiek ar divām akadēmiskajām nodarbībām, lai studenti jau spētu rakstīt pilnvērtīgas transformācijas valodā MOLA. Starp citām transformāciju valodām līdzīgākās pēc valodas stila valodai MOLA ir UMLX[39], AGG[44] un valodas MOF QVT[35] grafiskā valodas apakškopa. Tomēr katrā no tām ir kādi būtiski trūkumi. Piemēram, nevienā no tām nav labi atrisinātas valodas vadības struktūras, turklāt tajās ir arī samērā vāji un neērti paraugu (*pattern*) atbalsta risinājumi. Jāpiezīmē, ka būtisks trūkums daudzām modeļu transformāciju valodām ir arī nepietiekams atbilstošo rīku atbalsts, kam sekas ir apgrūtinoša šādu valodu praktiska lietošana. Savukārt valodai MOLA ir izstrādāts spēcīgs rīku atbalsts (skat. 4. nodaļu).

3.2 Valodas MOLA lietošanas metodoloģija un paraugu atpazīšanas efektivitāte

Šie pētījumi tika veikti paralēli ar pašas valodas MOLA izstrādi. Jau no paša sākuma, projektējot valodu MOLA, bija svarīgi domāt arī par valodas efektīvu implementāciju, lai neradītu neparedzētas problēmas vēlākā valodas realizācijā un pielietošanā reālos MDSD uzdevumos.

Pētījumu rezultātā ir izpētīta paraugu atpazīšanas efektivitāte (*pattern matching efficiency*) valodā MOLA. Tika izdalītas tās valodas MOLA sintakses un semantikas īpašības, kuras varētu būtiski ietekmēt MOLA transformāciju izpildes efektivitāti.

Lai varētu novērtēt paraugu atpazīšanas efektivitāti, tika izstrādāta hipotētiska valodas MOLA virtuālā mašīna. Virtuālā mašīna tika izstrādāta kā vienkāršu funkciju kopa, kura nodrošina elementāras darbības ar repozitoriju (datu glabātuvi). Zemāk ir pārskaitītas virtuālās mašīnas funkcijas, no kurām dažām ir precīzāk paskaidrota to nozīme:

- `getPatternRoot()` – atgriež parauga (*pattern*) saknes elementu (cikla mainīgo).

- `getPatternElement(int i)` – atgriež i-to parauga elementu.
- `getNext(metaClass mcl)` – funkcija atgriež nākamo klases `mcl` instanci. Gadījumā, ja vairs nav instances ko atgriezt, tad tiek atgriezta `null` konstante.
- `getNextByLink(association assoc, instance sourceInst, metaclass mcl)` – visbiežāk lietotā paraugu atpazīšanas funkcija. Funkcija secīgi atgriež klases `mcl` instances, kuras var tikt sasniegtas no fiksētas instances `sourceInst`, „staigājot" pa saitēm, kuras atbilst asociācijai `assoc`. Gadījumā, ja vairs nav instances ko atgriezt, tad tiek atgriezta `null` konstante. Šai funkcijai ir arī inicializācijas funkcija `initializeGetNextByLink` ar identiskiem parametriem.
- citas mazāk svarīgas virtuālās mašīnas funkcijas:
  - `eval(instance inst, oclExpression expr)`,
  - `checkLink(instance sourceInst, instance targInst, association assoc)`,
  - `getNextFromSet(metaClass mcl, set instSet)`,
  - `getNextByLink(association assoc, instance sourceInst, metaclass mcl)`,
  - `getNextByLinkFromSet(association assoc, instance sourceInst, metaclass mcl, set instSet)`,
  - un vēl dažas citas inicializācijas funkcijas.

Tika izstrādāts algoritms, kurš, izmantojot valodas MOLA hipotētiskās virtuālās mašīnas iespējas un ievērojot valodas programmēšanas īpatnības, realizēja paraugu atpazīšanu. Tika izstrādāts un piedāvāts "labais stils", kā programmēt valodā MOLA, lai paraugu atpazīšanas mehānisms strādātu efektīvi. „Labā stila" daži no pamatprincipiem bija – izmantot metamodeļa asociācijas ar kardinalitāti `0..1` vai `1` atbilstošajā virzienā (prom no cikla mainīgā) un saprātīgi izmantot references elementus gan iekļautajos ciklos, gan arī tur, kur asociāciju kardinalitātes nav `0..1` vai `1` atbilstošajā virzienā. Izstrādātā paraugu atpazīšanas algoritma sarežģītība tika novērtēta kā O(`n`), kur `n` ir ieejas modeļa izmērs (klašu instanču skaits). Novērtējums tika iegūts pie nosacījumiem, ka tiek ievērots piedāvātais programmēšanas stils valodā MOLA un uzdevumu principā ir iespējams algoritmiski atrisināt lineārā laikā.

Protams, vispārīgā gadījumā parauga atpazīšana var novest pie lielas elementu skaita pārlases. Tā tas varētu būt, ja MOLA programmu veido nesaprātīgi vai arī objektīvi ir jārealizē kāds uzdevums, kur piedāvātā stila programmēšanas principus nevar ievērot. Tādā nozīmē valoda MOLA ir līdzīga tradicionālām universālām programmēšanas valodām, kurās vienu un to pašu uzdevumu var algoritmiski implementēt dažādos veidos.

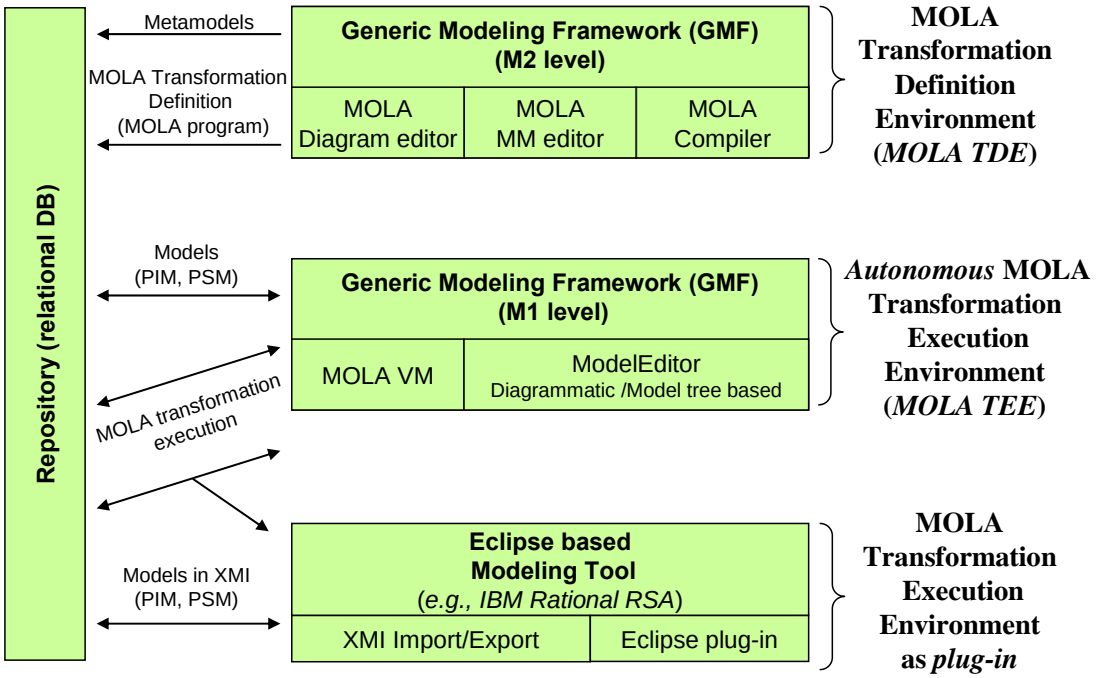
Lielākais autora ieguldījums šajā pētījumu posmā ir risinājumu izstrāde, kuri ir saistīti ar tādiem valodas MOLA aspektiem kā valodas MOLA **pamatkonstrukciju izstrāde**, **paraugu atpazīšanas efektivitāte** un **OO programmēšanas elementu ieviešana** valodā MOLA (piemēram, valodā tiek atbalstītas tādas klasiskas OO programmēšanas idejas kā mantošana un polimorfisms). Ļoti lielu ieguldījumu autors ir devis **hipotētiskās valodas MOLA virtuālās mašīnas** izstrādāšanā.

4 Valodas MOLA implementācija un valodas lietojumi

Šajā nodaļā ir apskatīti promocijas darba trešā pētījuma posma (2004.g.-2007.g.) rezultāti, kas ir apkopoti trīs publikācijās [11, 12, 13] un referēti trīs starptautiskās konferencēs. Autors ir aktīvi piedalījies visās ar šiem pētījumiem saistītās aktivitātēs. Šajā posmā veikto pētījumu rezultātā izstrādātais MOLA rīks (*MOLA Tool*) tika arī demonstrēts nozīmīgas starptautiskas konferences ("*European Conference on Model Driven Architecture – Foundations and Applications (ECMDA-FA)*") rīku demonstrācijas sesijā [20].

Lai izstrādāto valodu varētu ērti lietot modeļu transformāciju uzdevumos ir nepieciešama tās efektīva implementācija un atbilstošais rīku atbalsts. Šī laika posma pētījumu galvenais mērķis bija izstrādāt rīku komplektu, ar kura palīdzību būtu iespējams izveidot valodas MOLA transformācijas un tās izpildīt. Pētījumu galvenais praktiskais rezultāts ir izstrādātais MOLA rīks. MOLA rīks sastāv no vairākām būtiskām komponentēm (skatīt 4.1. attēlu):

- Valodas MOLA transformāciju definēšanas vide (***MOLA TDE*** – *MOLA transformation definition environment*).
- Autonomā valodas MOLA transformāciju izpildes vide (***MOLA TEE*** – *MOLA transformation execution environment*)
- Valodas MOLA spraudnis (***MOLA Eclipse Plug-in***)



4.1. attēls. MOLAS rīka implementācijas shēma.

MOLA TDE nodrošina redaktoru komplektu darbam ar metamodeļiem un valodas MOLA programmām, kā arī ar valodas MOLA kompilatoru. Par pamatu MOLA TDE tika izmantots universālais modelēšanas rīks. Vidē tiek nodrošināti arī dažādi citi

papildus servisi, piemēram, valodas MOLA atsevišķu procedūru imports/eksports xml formātā, kas nodrošina vairāku cilvēku vienlaicīgu darbu pie viena transformāciju projekta.

MOLA TEE galvenā sastāvdaļa ir valodas MOLA virtuālā mašīna. Tās funkcija ir nodrošināt transformāciju izpildi. Tā kā par izpildlaika repozitoriju (datu glabātuvi) tiek izmantota relāciju datu bāze, tad ir ļoti dabīgi un ērti valodas MOLA paraugu atpazīšanu realizēt kā valodas SQL vaicājumus.

MOLA spraudnis.
Lai varētu praktiski lietot valodu MOLA reālos MDSD uzdevumos vajadzēja realizēt iespēju to darbināt no izstrādes rīkiem, kuros tiek veikta modeļbāzēta sistēmu izstrāde. Tādēļ arī šajā laika posmā tika veikti intensīvi pētījumi par to, kā izmantot valodu MOLA kā spraudni citos modelēšanas rīkos. Pētījumu rezultātā tika izstrādātas vairākas komponentes, kuras nodrošina valodā MOLA rakstītu transformāciju izpildi modelēšanas rīkos, kuri ir būvēti izmantojot *Eclipse* platformu [67], piemēram, tādā firmas IBM rīkā kā *Rational Software Architect* [68]. Interesantākā no izstrādātajām komponentēm ir tā komponente, kura nodrošina universālu UML 2.0 XMI [69] importu/eksportu. Tā nodrošina datu importu/eksportu starp valodas MOLA izpildes vidi un tāda patvaļīga modelēšanas rīka vidi, kurš atbalsta minēto XMI standartu.

Šajā laika posmā tika veikti arī pētījumi par to, kā valodu MOLA varētu pielietot tipiskos MDSD uzdevumos. Tika izstrādātas un demonstrētas valodā MOLA izveidotās atbilstošās transformācijas. Piemēram, MOLA rīka demonstrācijā [20] tika rādīts, kā ir iespējams automātiski ar valodas MOLA transformāciju palīdzību IBM RSA rīkā izstrādātus modeļus ērti un viegli pārveidot (transformēt) no viena par otru. Konkrēti tika demonstrēts, kā no projektējuma UML klašu diagrammas var automātiski iegūt *Hibernate* platformai [70] ērti pielietojamu modeli. Tas ir klasisks PIM un PSM modeļu lietojums modeļbāzētā sistēmu izstrādē.

Lielākais autora ieguldījums šajā pētījumu posmā ir:

- Autora izprojektētais un izstrādātais **valodas MOLA spraudnis** IBM modelēšanas rīkam *Rational Software Architect*. Tas nodrošina valodā MOLA rakstītu transformāciju izpildi minētajā rīkā.
- Autora izprojektētā un izstrādātā universālā **UML 2.0 XMI importa/eksporta komponente**.
- Autora izprojektētais un izstrādātais **importa/eksporta serviss** MOLA rīkā. Tas nodrošina valodas MOLA procedūru importu/eksportu, kas ir būtisks serviss lielu transformāciju projektu izstrādē, kad projektā piedalās vienlaicīgi vairāki cilvēki.

5 Nobeigums

Promocijas darbs ir saistīts ar pētījumiem par universālu metamodeļbāzētu modelēšanas rīku būvi un modeļu transformāciju valodām un to pielietojumiem. Promocijas **darbā izvirzītie mērķi ir pilnībā izpildīti**. Zemāk ir īsi pārskaitīti pētījumu rezultāti:

- Izprojektēts un realizēts universālais modelēšanas **rīka prototips** (diagrammatiskais skatu attēlotājs) – jāpiezīmē, ka prototips vairs praktiski netiek lietots. Savulaik tieši ar šo rīka prototipa un redaktoru definēšanas valodu (EdDL) saistīto pētījumu rezultāti deva pārliecību un idejas par modeļu transformāciju nozīmīgumu rīku izstrādē.
- Izprojektēts un realizēts **universālais metamodeļbāzētais modelēšanas rīks** – rīks ir ļoti labi sevi pierādījis praksē un tas vēl joprojām aktīvi tiek izmantots dažādiem eksperimentiem MII pētījumu projektos. Izmantojot šo rīku, tika izveidots arī modeļu transformāciju valodas MOLA redaktors.
- Izstrādāta modeļu **transformāciju valoda MOLA** – unikāla grafiska modeļu transformāciju valoda. Tā ir jau praksē pārbaudīta un sekmīgi tiek lietota starptautiskos zinātniskos projektos, kā arī studentu apmācībā LU datorzinātņu maģistrantūrā.
- Izprojektēts un realizēts valodas **MOLA rīks –** nodrošina visus nepieciešamos servisus, lai valodā MOLA rakstītas transformācijas varētu pielietot modeļu transformāciju uzdevumos.
- Izstrādāta modeļu transformāciju **valodas MOLA tīmekļa vietne** [71].

Jāuzsver, ka ar transformāciju valodām saistītie **pētījumi tiek intensīvi turpināti**. Interesantākais darbs, kurā aktīvi piedalās arī promocijas darba autors, noris pie nākamās valodas MOLA un MOLA rīka versijas:

- Šobrīd MII tiek izstrādāta jaunas paaudzes uz metamodeļiem un modeļu transformācijām balstīta rīku platforma (*Transformation-based Tool Framework*) [22], kurā modeļu transformācijas tiek būvētas esošajā MOLA valodā. Šajā platformā tiek izstrādāts arī nākamais valodas MOLA rīks. Respektīvi, nākamā valodas MOLA versija tiek realizēta, izmantojot esošo valodas MOLA versiju, pielietojot tā saucamo „*bootstrapping*" [72] metodi.
- Interesantākā pētījumu tēma, kura ir saistīta ar nākamo valodas MOLA versiju, ir pētījumi par to, kā valodā MOLA varētu ieviest tādus OO programmēšanas līdzekļus kā šablonus (saukti par *template* valodā C++ vai *generic* valodā Java).

6 Atsauces

6.1 Autora publikācijas recenzētos starptautisku konferenču materiālos

1. A. Kalnins, K. Podnieks, A. Zarins, E. Celms , J. Barzdins. ***Editor definition language and its implementation.*** - Lecture Notes in Computer Science, Springer, v. 2244, 2001, pp.530-537.
2. A. Kalnins, J. Barzdins, E. Celms et al. ***The first step towards generic modeling tool.*** - Proceedings of the Fifth International Baltic Conference on Databases and Information Systems, Tallin, 2002, v.2, pp.167-180.
3. L. Lace, E. Celms, A. Kalnins. ***Diagram definition facilities in a generic modeling tool.*** - Proceedings of International Conference on Modelling and Simulation of Business systems, Vilnius, 2003, pp.220-224.
4. E. Celms, A. Kalnins, L. Lace. ***Diagram definition facilities based on metamodel mappings.*** - Proceedings of the 3rd OOPSLA (Workshop on Domain-Specific Modeling) , University of Jyvaskyla, 2003, pp.23-32.
5. Celms E. ***Generic Data Representation by Table in Metamodel Based Modelling Tool.*** Scientific proceedings of University of Latvia, Computer Science and Information Technologies, Automation of Information Processing, vol. 669, Riga, Latvia, April 2004, pp. 53-61.
6. A. Kalnins, J. Barzdins, E. Celms. ***Model Transformation Language MOLA.*** - Lecture Notes in Computer Science, Springer, v. 3599, 2005. Model Driven Architecture: European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004, Twente, The Netherlands, June 26-27, 2003 and Linkoping, Sweden, June 10-11, 2004. Revised Selected Papers, pp. 62-76.
7. A. Kalnins, J. Barzdins, E. Celms. ***Basics of Model Transformation Language MOLA.*** - ECOOP 2004 (Workshop on Model Transformation and execution in the context of MDA) , Oslo, Norway, June 14-18, 2004, p. 6.
8. A. Kalnins, J. Barzdins, E. Celms. ***Model Transformation Language MOLA: Extended Patterns.*** - Databases and Information Systems, Selected papers from the 6th International Baltic Conference DB&IS'2004, IOS Press, FAIA (Frontiers in Artificial Intelligence and Applications), vol. 118, 2005, pp. 169-184.
9. A. Kalnins, J. Barzdins, E. Celms. ***MOLA Language: Methodology Sketch.*** - Proceedings of EWMDA-2, Canterbury, England, September 7-8, 2004, pp.194-203.
10. A. Kalnins, J. Barzdins, E. Celms. ***Efficiency Problems in MOLA Implementation.*** 19th International Conference, OOPSLA'2004 (Workshop "Best Practices for Model-Driven Software Development") , Vancouver, Canada, October 2004, p. 14.
11. A. Kalnins, E. Celms, A. Sostaks. ***Tool support for MOLA.*** Fourth International Conference on Generative Programming and Component Engineering (GPCE'05). Proceedings of the Workshop on Graph and Model Transformation (GraMoT) , Tallinn, Estonia, September 2005, pp. 162-173.
12. A. Kalnins, E. Celms, A. Sostaks. ***Model Transformation Approach Based on MOLA.*** ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML '2005). (MoDELS/UML'05 Workshop: Model Transformations in Practice (MTIP)) , Montego Bay, Jamaica, October 2-7, 2005, p. 25.
13. A. Kalnins, E. Celms, A. Sostaks. ***Simple and Efficient Implementation of Pattern Matching in MOLA Tool***. Proceedings of the 7th International Baltic Conference on Databases and Information Systems (Baltic DB&IS'2006). , Vilnius, Lithuania, July 3-6, 2006, pp. 159-167.

6.2 Citas autora publikācijas recenzētos starptautisku konferenču materiālos (tieši nesaistītas ar promocijas darba tēmu)

14. A. Kalnins, J. Barzdins, E. Celms. ***UML Business Modeling Profile***. - Proceedings of ISD'2004, Vilnius, Lithuania, September 9-11, 2004, pp.182-194.
15. J.Viksna, E.Celms, M.Opmanis, K.Podnieks, P.Rucevskis, A.Zarins, A.Barrett, S.Guha Neogi, M.Krestyaninova, M.McCarthy, A.Brazma, U.Sarkans. ***PASSIM - an open source software system for managing information in biomedical studies***. BMC Bioinformatics, vol. 8:52, 2007.

6.3 Citi promocijas darbā izmantotie avoti

16. ***Domain-specific language, DSL.***
    Internet – http://en.wikipedia.org/wiki/ Domain-specific_programming_language
17. Stahl Thomas, Volter Markus. ***Model-Driven Software Development***. John Wiley & Sons, Ltd., 2006.
18. Jorn Bettin. ***Model-Driven Software Development: An emerging paradigm for industrialized software asset development.*** 2004. Internet – http://www.softmetaware.com/mdsd-and-isad.pdf.
19. ***Object Management Group (OMG).*** Internet – http://www.omg.org
20. Tool session of the "***European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA)***", November 7-10th, 2005, Nuernberg, Germany.
21. ***ReDSeeDS (Requirements-Driven Software Development System).***
    Internet – http://www.redseeds.eu
22. A. Kalnins, J. Barzdins. ***MDA Support by Transformation Based Tool.*** Proceedings of First International Workshop MoRSe 2006, Warsaw, Poland, October 2006, pp. 21-24.
23. Object Management Group. ***Meta Object Facility (MOF).*** Internet – http://www.omg.org/mof
24. Object Management Group. ***Meta Object Facility Core Specification***, version 2.0, 2006.
    Internet – http://www.omg.org/docs/formal/06-01-01.pdf
25. Booch G., Jackobson I., Rumbaugh J. ***The Unified Modeling Language. Reference Manual***, Addison-Wesley, 1999..
26. Object Management Group. ***Unified Modeling Language: Superstructure.*** Version 2.0 (Final Adopted Specification), 2005. Internet – http://www.omg.org/docs/formal/05-07-04.pdf
27. Object Management Group. ***Unified Modeling Language: Infrastructure***. Version 2.0 (Final Adopted Specification), 2005. Internet – http://www.omg.org/docs/formal/05-07-05.pdf
28. Object Management Group. ***Unified Modeling Language: Superstructure.*** Version 2.1.1, 2007.
    Internet – http://www.omg.org/docs/formal/07-02-05.pdf
29. Object Management Group. ***Unified Modeling Language: Infrastructure.*** Version 2.1.1, 2007.
    Internet – http://www.omg.org/docs/formal/07-02-06.pdf
30. Object Management Group. ***Business Process Modeling Notation (BPMN)***, Final Adopted Specification, OMG, 2006. Internet – http://www.omg.org/docs/dtch/06-02-01.pdf
31. Object Management Group. ***Model Driven Architecture (MDA).***
    Internet – http://www.omg.org/mda
32. Kleppe A., Warmer J., Bast W. ***MDA Explained. The model driven architecture: practice and promise.*** Addison-Wesley, 2003.
33. Object Management Group. ***MDA Guide Version 1.0.1.***
    Internet– http://www.omg.org/docs/omg/03-06-01.pdf
34. Object Management Group. ***Request for Proposal: MOF 2.0 Query / Views / Transformations***, 2002. Internet – http://www.omg.org/docs/ad/02-04-10.pdf
35. Object Management Group. ***QVT – Query/View/Transformation Specification.*** (Final Adopted Specification), 2005. Internet – www.omg.org/docs/ptc/05-11-01.pdf
36. IMCS (LUMII). ***The Base Transformation Language L0***, 2007.
    Internet – http://l0.mii.lu.lv/L0_plus_CurrVers_2_4.pdf
37. Agrawal A., Karsai G, Shi F. ***Graph Transformations on Domain-Specific Models***. Technical report, Institute for Software Integrated Systems, Vanderbilt University, ISIS-03-403, 2003
38. Vanderbilt University. **GReAT**. Internet – http://repo.isis.vanderbilt.edu/tools/get_tool?GReAT
39. Willink E.D. ***A concrete UML-based graphical transformation syntax - The UML to RDBMS example in UMLX***. Workshop on Metamodelling for MDA, University of York, England, 24-25 November, 2003.
40. Bezivin J., Dupe G., Jouault F., et al. ***First experiments with the ATL model transformation language: Transforming XSLT into XQuery***. 2nd OOPSLA Workshop on Generative Techniques in Context of MDA, Anaheim, California, 2003.
41. Institut national de recherche en informatique et en automatique (INRIA). ***ATL : Atlas Transformation Language***. Internet – http://www.sciences.univ-nantes.fr/lina/atl
42. ***Simple Transformation Rule Language (TRL).*** Internet – http://modfact.lip6.fr/qvtP.html
43. DSTC. ***Tefkat: The EMF Transformation Engine.***Online documentation.
    Internet – http://www.dstc.edu.au/tefkat
44. TU Berlin, TFS. ***The Attributed Graph Grammar System (AGG).***
    Internet – http://tfs.cs.tu-berlin.de/agg

45. IBM. ***Model Transformation Framework (MTF)***.
Internet – http://www.alphaworks.ibm.com/tech/mtf
46. McGill University, Modelling, Simulation and Design Lab. ***ATOM***.
Internet – http://atom3.cs.mcgill.ca
47. Budapest University of Technology and Economics, Department of Automation and Applied Informatics. ***Visual Modeling and Transformation System (VMTS).***
Internet – http://avalon.aut.bme.hu/~tihamer/research/vmts
48. Institut fur Informatik der Technischen Universitat Munchen, Peter Braun, Frank Marschall. ***The Bidirectional Object Oriented Transformation Language (BOTL).***
Internet – http://wwwbib.informatik.tu-muenchen.de/infberichte/2003/TUM-I0307.pdf
49. Octavian Patrascoiu. ***Yet Another Transformation Language (YATL).*** Proceedings of the 1st European MDA Workshop, MDA-IA, pages 83-90. University of Twente, the Nederlands, January 2004. Internet – http://www.cs.kent.ac.uk/pubs/2004/1829
50. Universitat Paderborn, Institut fur Informatik. ***Fujaba.***
Internet – http://wwwcs.uni-paderborn.de/cs/fujaba/documents/user/manuals/FujabaDoc.pdf
51. Budapest University of Technology and Economics, GMT subproject. ***Visual Automated Model Transformations (VIATRA2).***
Internet – http://dev.eclipse.org/viewcvs/indextech.cgi/gmt-home/subprojects/VIATRA2/index.html
52. Jesus Sanchez Cuadrado, Jesus Garcia Molina, Marcos Menarguez Tortosa. ***RubyTL: A Practical, Extensible Transformation Language.*** Model Driven Architecture - Foundations and Applications, Second European Conference, ECMDA-FA 2006, Bilbao, Spain, July 10-13, 2006. LNCS 4066, Springer 2006.
53. Kleppe, A. ***Towards general purpose high level software languages***. Model Driven Architecture - Foundations and Applications (A. Hartman and D. Kreische, eds.), vol. 3748 of LNCS, Springer-Verlag, Nov. 2005.
54. Laurence Tratt. ***The MT model transformation language***. Technical report TR-05-02, Department of Computer Science, King's College London, May 2005.
55. Joint revised submission by Compuware Corporation, SUN Microsystems, … (OMG Document ad/2003-08-07). ***XMOF Queries, Views and Transformations on Models using MOF, OCL and Patterns***, 2003. Internet – http://www.omg.org/docs/ad/03-08-07.pdf
56. Compuware. ***OptimalJ (Model-driven development for java).***
Internet – http://www.compuware.com/products/optimalj/
57. EU ESPRIT project, ***Application Development for the Distributed Enterprise (ADDE)***.
Internet – http://www.fast.de/ADDE
58. Smolander, K., Martiin, P., Lyytinen, K., Tahvanainen, V-P. ***Metaedit – a flexible graphical environment for methodology modelling.*** Springer-Verlag, 1991.
59. Ebert, J., Suttenbach, R., Uhe, I. ***Meta-CASE in Practice: a Case for KOGGE***. Proceedings of the 9th International Conference, CAiSE'97, Barcelona, Catalonia, Spain , 1997, pp.203-216.
60. Lincoln Software Ltd. ***IPSYS Toolbuilder Manual***, Version. 2.1, 1996.
61. Honeywell Inc. ***The Domain Modeling Environment (DOME),*** Users Guide.
Internet – http://www.htc.honeywell.com/dome
62. ETH Zurich, TIK. ***The Moses project.*** Internet – http://www.tik.ee.ethz.ch/~moses
63. Object Management Group. ***OCL 2.0 Specification***, Version 2.0, 2006.
Internet – http://www.omg.org/docs/formal/06-05-01.pdf
64. MetaCase. ***MetaEdit+ resources.*** Internet – http://www.metacase.com/papers/index.html
65. Ledeczi, A., Maroti, M., Bakay, A., Karsai, G., Garrett, J., Thomason IV, C., Nordstrom, G., Sprinkle, J., Volgyesi, P. ***The Generic Modeling Environment (GME)***. Workshop on Intelligent Signal Processing, Budapest, Hungary, May 17, 2001.
66. de Lara, J,, Vangheluwe, H., Alfonseca, M. ***Meta-Modelling and Graph Grammars for Multi-Paradigm Modelling in AToM3***. Software and Systems Modeling (SoSyM), Volume 3, Number 3, August 2004, pp. 194-209.
67. ***Eclipse***. Internet – http://www.eclipse.org
68. IBM. ***Rational Software Architect (RSA).***
Internet – http://www-306.ibm.com/software/awdtools/architect/swarchitect/index.html
69. Object Management Group. ***MOF 2.0/XMI Mapping Specification***, v2.1, 2005.
Internet – http://www.omg.org/docs/formal/05-09-01.pdf
70. ***Hibernate***. Internet – http://www.hibernate.org
71. IMCS (LUMII). ***MOLA home page***. Internet – http://mola.mii.lu.lv
72. ***Bootstrapping.*** Internet – http://en.wikipedia.org/wiki/Bootstrapping_%28computing%29

7 Pielikums

7.1 Autora referāti par darba rezultātiem starptautiskās zinātniskās konferencēs vai semināros

1. L. Lace, E. Celms, A. Kalnins. ***Diagram definition facilities in a generic modeling tool.*** - International Conference on Modelling and Simulation of Business systems, Vilnius, 2003, pp.220-224.

2. A. Kalnins, J. Barzdins, E. Celms. ***Model Transformation Language MOLA.*** - Model Driven Architecture: European MDA Workshop: Foundations and Applications, MDAFA 2004, Linkoping, Sweden, June 10-11, 2004.

3. A. Kalnins, J. Barzdins, E. Celms. ***Basics of Model Transformation Language MOLA.*** - ECOOP 2004 (Workshop on Model Transformation and execution in the context of MDA) , Oslo, Norway, June 14-18, 2004.

4. A. Kalnins, J. Barzdins, E. Celms. ***Model Transformation Language MOLA: Extended Patterns.*** - Databases and Information Systems, 6th International Baltic Conference DB&IS'2004, Riga, Latvia, 2004.

5. A. Kalnins, J. Barzdins, E. Celms. ***Efficiency Problems in MOLA Implementation.*** 19th International Conference, OOPSLA'2004 (Workshop "Best Practices for Model-Driven Software Development") , Vancouver, Canada, October 2004.

6. A. Kalnins, E. Celms, A. Sostaks. ***Tool support for MOLA.*** Fourth International Conference on Generative Programming and Component Engineering (GPCE'05). Workshop on Graph and Model Transformation (GraMoT) , Tallinn, Estonia, September 2005.

7. A. Kalnins, E. Celms, A. Sostaks. ***Model Transformation Approach Based on MOLA.*** ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML '2005). (MoDELS/UML'05 Workshop: Model Transformations in Practice (MTIP)) , Montego Bay, Jamaica, October 2-7, 2005.

7.2 Promocijas darbā iekļautās publikācijas un promocijas darba autora personiskais ieguldījums.

Autori	Publikācija	Promocijas darba autora ieguldījums procentos no kopējā pētījumu apjoma	Promocijas darba autora ieguldījuma apraksts
A. Kalniņš, K. Podnieks, A. Zariņš, E. Celms, J. Bārzdiņš.	***Editor definition language and its implementation.*** - Lecture Notes in Computer Science,	30%	• Piedalīšanās ideju izstrādē. • EdDL valodas pamatkonstrukciju izstrāde. • Anotāciju kompilētāja un redaktoru dzinēja

	Springer, v. 2244, 2001, pp.530-537.		projektēšana un izstrāde.
A. Kalniņš, J. Bārzdiņš, E. Celms, L. Lāce, M. Opmanis, K. Podnieks, A. Zariņš.	***The first step towards generic modeling tool.*** - Proceedings of the Fifth International Baltic Conference on Databases and Information Systems, Tallin, 2002, v.2, pp.167-180.	30%	• Piedalīšanās ideju izstrādē. • Rūpnieciskā universālā metamodeļbāzētā modelēšanas rīka arhitektūras pamatprincipu izstrāde. • Universālā modelēšanas rīka dispečera projektēšana un realizācija. • Modeļu koka pārvaldības dzinēja projektēšana un realizācija.
L. Lāce, E. Celms, A. Kalniņš.	***Diagram definition facilities in a generic modeling tool.*** - Proceedings of International Conference on Modelling and Simulation of Business systems, Vilnius, 2003, pp.220-224.	40%	• Piedalīšanās ideju izstrādē. • Diagrammu anotāciju definēšanas (attēlojuma) pamatprincipu izstrāde. • Diagrammu anotāciju to atbalsts universālā metamodeļbāzētā modelēšanas rīkā.
E. Celms, A. Kalniņš, L. Lāce.	***Diagram definition facilities based on metamodel mappings.*** - Proceedings of the 3rd OOPSLA (Workshop on Domain-Specific Modeling) , University of Jyvaskyla, 2003, pp.23-32.	50%	• Piedalīšanās ideju izstrādē. • Attēlojumu tipu pamatprincipu izstrāde. • Attēlojumu tipu bibliotēkas izstrāde.
E. Celms	***Generic Data Representation by Table in Metamodel Based Modelling Tool.*** Scientific proceedings of University of Latvia, Computer Science and Information Technologies, Automation of Information Processing, vol. 669, Riga, Latvia, April 2004, pp. 53-61.	100%	• Piedalīšanās ideju izstrādē. • Universālā tabulārā skatu dzinēja projektēšana un realizācija. • Universālā īpašību redaktoru dzinēja projektēšana un realizācija.
A. Kalniņš, J. Bārzdiņš, E. Celms.	***Model Transformation Language MOLA.*** - Lecture Notes in Computer Science,	40%	• Piedalīšanās ideju izstrādē. • Modeļu transformāciju valodas MOLA

	Springer, v. 3599, 2005. Model Driven Architecture: European MDA Workshops: Foundations and Applications, MDAFA 2003 and MDAFA 2004, Twente, The Netherlands, June 26-27, 2003 and Linkoping, Sweden, June 10-11, 2004. Revised Selected Papers, pp. 62-76.		pamatkonstrukciju izstrāde.
A. Kalniņš, J. Bārzdiņš, E. Celms.	***Basics of Model Transformation Language MOLA.*** - ECOOP 2004 (Workshop on Model Transformation and execution in the context of MDA) , Oslo, Norway, June 14-18, 2004, p. 6.	60%	• Piedalīšanās ideju izstrādē. • OO programmēšanas elementu ieviešana valodā MOLA.
A. Kalniņš, J. Bārzdiņš, E. Celms.	***Model Transformation Language MOLA: Extended Patterns.*** - Databases and Information Systems, Selected papers from the 6th International Baltic Conference DB&IS'2004, IOS Press, FAIA (Frontiers in Artificial Intelligence and Applications), vol. 118, 2005, pp. 169-184.	60%	• Piedalīšanās ideju izstrādē. • Paplašināto paraugu pamatprincipu izstrāde.
A. Kalniņš, J. Bārzdiņš, E. Celms.	***MOLA Language: Methodology Sketch.*** - Proceedings of EWMDA-2, Canterbury, England, September 7-8, 2004, pp.194-203.	60%	• Piedalīšanās ideju izstrādē. • Transformāciju programmēšanas metodikas izstrādāšana valodai MOLA.
A. Kalniņš, J. Bārzdiņš, E. Celms.	***Efficiency Problems in MOLA Implementation.*** 19th International Conference, OOPSLA'2004 (Workshop "Best Practices for Model-	80%	• Piedalīšanās ideju izstrādē. • Valodas MOLA hipotētiskās virtuālās mašīnas izstrādāšana. • Paraugu atpazīšanas efektivitātes novērtējumi valodai MOLA.

	Driven Software Development") , Vancouver, Canada, October 2004, p. 14.		• Valodas lietošanas pamatprincipu izstrādāšana, kuri nodrošina valodas vieglu implementēšanu.
A. Kalniņš, E. Celms, A. Šostaks.	***Tool support for MOLA.*** Fourth International Conference on Generative Programming and Component Engineering (GPCE'05). Proceedings of the Workshop on Graph and Model Transformation (GraMoT) , Tallinn, Estonia, September 2005, pp. 162-173.	50%	• Piedalīšanās ideju izstrādē. • MOLA rīka arhitektūras pamatprincipu izstrāde. • Universālu UML 2.0 XMI importa/ eksporta pamatprincipu izstrādāšana un realizēšana. • MOLA rīka Eclipse ietvara spraudņa projektēšana un izstrāde.
A. Kalniņš, E. Celms, A. Šostaks.	***Model Transformation Approach Based on MOLA.*** ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS/UML '2005). (MoDELS/UML'05 Workshop: Model Transformations in Practice (MTIP)) , Montego Bay, Jamaica, October 2-7, 2005, p. 25.	40%	• Piedalīšanās ideju izstrādē. • Rekursijas lietošanas pamatprincipu izstrāde valodai MOLA.
A. Kalniņš, E. Celms, A. Šostaks.	***Simple and Efficient Implementation of Pattern Matching in MOLA Tool***. Proceedings of the 7th International Baltic Conference on Databases and Information Systems (Baltic DB&IS'2006). , Vilnius, Lithuania, July 3-6, 2006, pp. 159-167.	20%	• Piedalīšanās ideju izstrādē. • Importa/eksporta servisu projektēšana un izstrāde MOLA rīkā.