RĪGAS TEHNISKĀ UNIVERSITĀTE
Datorzinātnes un informācijas tehnoloģijas fakultāte
Lietišķo datorsistēmu institūts

Uldis DONIŅŠ
Doktora studiju programmas „Datorsistēmas" doktorants

TOPOLOĢISKĀ VIENOTĀ MODELĒŠANAS VALODA: IZSTRĀDE UN LIETOŠANA

Promocijas darba kopsavilkums

Zinātniskais vadītājs
Dr.habil.sc.ing., profesors
J. OSIS

Rīga 2012

UDK 004.414.23 (043.2)
Do 515 t

Doniņš U. Topoloģiskā vienotā modelēšanas valoda: izstrāde un lietošana. Promocijas darba kopsavilkums. - R.:RTU, 2012. - 36 lpp.

Iespiests saskaņā ar Rīgas Tehniskās universitātes Datorzinātnes un informācijas tehnoloģijas fakultātes Lietišķo datorsistēmu institūta padomes 2012. gada 26. jūnija lēmumu Nr. 77.

Šis darbs izstrādāts ar Eiropas Sociālā fonda atbalstu projektā «Atbalsts RTU doktora studiju īstenošanai».

ISBN 978-9934-10-352-0

PROMOCIJAS DARBS
IZVIRZĪTS INŽENIERZINĀTŅU DATORSISTĒMU DOKTORA GRĀDA IEGŪŠANAI RĪGAS TEHNISKAJĀ UNIVERSITĀTĒ

Promocijas darbs inženierzinātņu datorsistēmu doktora grāda iegūšanai tiek publiski aizstāvēts 2012. g. 15. oktobrī Rīgas Tehniskās universitātes Datorzinātnes un informācijas tehnoloģijas fakultātē, Meža ielā 1, 3. korpusā 202. auditorijā.

OFICIĀLIE RECENZENTI:

Profesors, Dr.habil.sc.ing. Jānis Grundspeņķis
Rīgas Tehniskā universitāte, Rīga, Latvija

Profesors, Dr.habil.sc.comp. Jānis Bārzdiņš
Latvijas universitāte, Rīga, Latvija

Profesors, Ph.D. Lešeks Maciašeks (Leszek Maciaszek)
Makkvarija universitāte (Macquarie University), Sidneja, Austrālija

APSTIPRINĀJUMS

Apstiprinu, ka esmu izstrādājis doto promocijas darbu, kas iesniegts izskatīšanai Rīgas Tehniskajā universitātē inženierzinātņu doktora grāda iegūšanai. Promocijas darbs nav iesniegts nevienā citā universitātē zinātniskā grāda iegūšanai.

Uldis Doniņš ……. (paraksts)

Datums: ……

Promocijas darbs ir uzrakstīts angļu valodā, satur ievadu, 5 nodaļas, secinājumus, literatūras sarakstu, 14 pielikumus, 71 zīmējumu un ilustrāciju, 32 tabulas, kopā 224 lappuses. Literatūras sarakstā ir 134 nosaukumi.

SATURS

VISPĀRĪGS PROMOCIJAS DARBA RAKSTUROJUMS

Liela nozīme, izstrādājot programmatūru, ir problēmvides analīzei un nepieciešamā risinājuma jeb lietojumvides projektēšanai. Neskatoties uz to, ka programmatūras izstrādes kopienā tiek lietoti dažādi rīki, pieejas un metodes detalizēta risinājuma projektējuma sagatavošanai, problēmvides funkcionēšanas analīze tiek veikta nepietiekamā līmenī vai izlaista vispār. Viena no šādām pieejām ir objektorientēta programmatūras izstrāde, kas nosaka divus fundamentālus aspektus sistēmu modelēšanā – analīzi un projektēšanu [9]. Analīze definē risinājuma veicamos uzdevumus problēmvidē, lai tas atbilstu klienta izvirzītajām programmatūras prasībām. Savukārt projektējums nosaka to, kādā veidā programmatūras sistēma tiks izstrādāta. Objektorientēto sistēmu projektēšanu pēdējās desmitgades laikā nosaka un virza vienotā modelēšanas valoda UML (*Unified Modeling Language*) [14]. UML valoda ir programmatūras izstrādes industrijas apstiprināta standarta modelēšanas notācija, kas paredzēta programmatūras intensīvu sistēmu artefaktu vizualizēšanai, specificēšanai, konstruēšanai un dokumentēšanai [57]. Neskatoties uz UML valodā esošajiem elementiem programmatūras sistēmas artefaktu projektēšanai un specificēšanai, tā neietver iespējas problēmvides funkcionēšanas dokumentēšanai, lietojot no skaitļošanas neatkarīgus elementus. Tā kā UML valoda ir notācija, nevis tehnika vai metode, tās lietošanu programmatūras analīzei un projektēšanai vada un nosaka dažādas programmatūras izstrādes metodes un pieejas.

Pētījuma motivācija

Neskatoties uz to, ka pastāv vairākas programmatūras modelēšanas valodas (ieskaitot UML valodu, kuru ir apstiprinājusi un publicējusi Object Management Group (OMG) organizācija) un metodes, kas lieto šīs modelēšanas valodas, programmatūras izstrādes līmenis joprojām saglabājas pietiekami zems (t.i., lielu programmatūru izstrādes tiek atceltas, pārtērē atvēlēto budžetu un laika plānu, kā arī nereti izlaistā programmatūra ir zemā kvalitātes līmenī) [37]. Šis fakts rodas no tā, ka nereti problēmvide pastāv un funkcionē atrauti no sagatavotā risinājuma, jo programmatūras analīzes laikā netiek veltīta pienācīga uzmanība problēmvides funkcionēšanai [64]. Atsevišķos gadījumos programmatūras funkcionēšana tiek sagatavota pēc izstrādātāju ieskatiem, nevis atbilstoši problēmvides funkcionēšanas īpašībām. Samazinot vai pat pilnībā izlaižot pienācīgu problēmvides analīzi, nevar tikt izveidotas trasējamības saites starp problēmvides un risinājuma vides artefaktiem. Bez šīm trasējamības saitēm izstrādātās programmatūras akcepttestēšana zaudē savu jēgu, jo pasūtītājam nav iespējams pilnībā pārbaudīt piegādāto risinājumu [6]. Lai motivētu programmatūras izstrādātājus pievērst lielāku uzmanību problēmvides funkcionēšanai un tās analīzei, jānodrošina piemēroti programmatūras specificēšanas modeļi un to izstrādes metode. Džonsa (*Jones*) veiktā pētījuma ([37]) rezultāti parāda, ka UML valoda un pašlaik pieejamās modelēšanas metodes nenodrošina šādus piemērotus līdzekļus.

Pētījumu apgabals

Pētījuma apgabals ir sistēmu funkcionēšanas topoloģiskā modelēšana, kura tika izveidota 1960-to gadu vidū Rīgas Tehniskajā universitātē Jāņa Oša vadībā. Pirmie Topoloģiskā funkcionēšanas modeļa (TFM) teorētiskie pamati un tā lietošana ir publicēti [63]. Sākotnējā TFM lietošanas problēmvide ir mehānisko iekārtu diagnostika (piemēram, iekšdedzes dzinēju), kas tiek balstīta uz kibernētiku un datorzinātni. Liels skaits ar augstas kvalitātes algoritmiem un metodēm TFM lietošanai diagnostikas uzdevumu risināšanā ir apkopotas [70], savukārt [62] un [63] piedāvā jaunus teorētiskos pamatus sistēmu teorijai. TFM lietošana dažādās problēmvidēs tiek attīstīta joprojām.

Sistēmu funkcionēšanas topoloģiskā modelēšana tiek veiksmīgi izmantota medicīnas problēmu risināšanā, diagnostikā un ekspertu lēmumu pieņemšanā Zigurda Markoviča vadībā [38][48]. Pētījumu virziens, kuru turpina Jānis Grundspeņķis, sākotnēji ir saistīts ar ciklu hierarhiju analīzi racionāla diagnostikas algoritma izstrādei [29], [33]. Vēlāk Jāņa Grundspeņķa pētījumu virziens tiek attīstīts kā struktūrmodelēšana [32], kas tiek attīstīta sistemātiskai zināšanu iegūšanai, lietojot cēloņseku domēna modeli. Struktūrmodelēšanas būtība ir sistemātiska procedūra trīs struktūras (t.i. topoloģisko) modeļu konstruēšanai, kuri attēlo sarežģītu tehnisku sistēmu morfoloģiju, funkcijas un uzvedību [30]. Šī pieeja ir implementēta automatizētas struktūrmodelēšanas sistēmā ASMOS [31].

Jāņa Oša turpinātais pētījumu virziens ir saistīts ar TFM lietošanu objektorientētu sistēmu analīzei un izstrādei. Pēdējie publicētie pētījumu rezultāti ietver topoloģiskās modelēšanas lietošanu biznesa procesu modelēšanai un simulēšanai [73], TFM izmantošanu mehatronisku un iegulto sistēmu programmatūras izstrādē [60], problēmvides analīzes formālisma līmeņa paaugstināšana ([4], [5], [16] un [18]), no skaitļošanas neatkarīgā modeļa (*Computation Independent Model*, CIM) formāla analīze modeļvadāmās arhitektūras kontekstā (*Model Driven Architecture*, MDA [51]) ([61], [64], [68], [85] un [86]), platformneatkarīgā modeļa (*Platform Independent Model*, PIM) formāla specificēšana modeļvadāmajā arhitektūrā ([19], [21]) un iegulto sistēmu analīze un projektēšana, lietojot topoloģisko funkciju un arhitektūras kopprojektēšanas metodi [71]. TFM teorētiskie pamati ir apkopoti un publicēti monogrāfijā [65], sniedzot TFM un tā elementu definīcijas un parādot tā iespējas un priekšrocības formālā problēmvides analīzē. Šis pētījums ir sistēmu funkcionēšanas topoloģiskās modelēšanas attīstīšana objektorientētajā sistēmanalīzē un programmatūras izstrādē.

Darba mērķis un uzdevumi

Darba mērķis ir papildināt UML valodu ar teorētisko bāzi, lai radītu pamatu notācijas pārvēršanai par formālu modelēšanas valodu, un sagatavot modelēšanas metodi, kas ļauj viennozīmīgi izsekot cēloņseku sakarībām kā problēmvidē, tā lietojumvidē.

Darba uzdevumi izvirzītā mērķa sasniegšanai ir šādi:

1. Izpētīt UML valodas un tās specifikācijas attīstību, lai identificētu pozitīvos un negatīvos aspektus pašreizējās valodas versijas lietošanai programmatūras izstrādes laikā, kā arī noskaidrot pilnveidojamos valodas aspektus,
2. Identificēt UML valodas paplašināšanas mehānismus un iespējas, lai noteiktu piemērotāko veidu jaunas UML valodas versijas izstrādei,
3. Analizēt UML valodas virzīto modelēšanas metožu galvenās īpašības un salīdzināt to iespējas problēmvides formalizēšanai un risinājuma projektēšanai atbilstoši problēmvides funkcionēšanas īpašībām,
4. Izstrādāt UML valodas profila specificēšanas šablonu, veicot esošo profilu un to specifikāciju analīzi,
5. Specificēt jaunu modelēšanas valodu – Topoloģisko vienoto modelēšanas valodu (saīsināti TopUML) – atbilstoši identificētajiem UML valodas pilnveidojamajiem aspektiem un sagatavotajam profilu specifikācijas šablonam,
6. Sagatavot sistēmanalīzes un programmatūras projektēšanas metodi, kas atbalsta formālu TopUML profila lietošanu, tādejādi ļaujot viennozīmīgi izsekot problēmvides un lietojumvides cēloņseku sakarībām,
7. Aprobēt izstrādāto modelēšanas valodu un tās lietošanas metodi eksperimentālā problēmvides analīzes un programmatūras projektēšanas projektā, iesaistot procesā arī vairākas programmatūras izstrādes ekspertu grupas,
8. Aprobēt izstrādāto modelēšanas valodu un tās lietošanas metodi reālā programmatūras izstrādes projektā, veicot detalizētu procesa un sagatavoto artefaktu analīzi un fiksēšanu.

Pētījuma objekts un priekšmets

Pētījuma objekts ir UML valoda un tās lietošanas metodes programmatūras projektēšanas un izstrādes procesā.

Pētījuma priekšmets ir UML valodas un tās lietošanas metodes, fokusējoties uz formāla programmatūras projektējuma izstrādi un cēloņseku attiecību nodrošināšanu problēmvides un lietojumvides artefaktos.

Pētījumu metodes

Problēmvides un lietojumvides specificēšanai tiek lietots matemātisks modelis un modelēšanas valoda. Tāpat tiek izmantota metamodelēšanas metode, modeļu transformācijas, kā arī tādas matemātikas nodaļas kā vispārējā topoloģija, kombinatorā topoloģija, grafu teorija un matemātiskā loģika.

Zinātniskā novitāte un praktiskā vērtība

Pētījuma **zinātniskā novitāte** ir formāla lietojumvides modelēšana, atbilstoši problēmvides funkcionēšanai, kas tiek panākta lietojot TopUML profilu un tā lietošanas metodi programmatūras analīzei un projektēšanai. TopUML modelēšanas metode ir izstrādāta tā, lai vadītu un nodrošinātu atbilstošu TopUML diagrammu konstruēšanu un transformēšanu.

TopUML modelēšanas metodes īpašības ir salīdzinātas ar pašreiz esošajām UML valodas vadītām programmatūras izstrādes metodēm un tehnikām.

Pētījuma **praktiskā vērtība** ir TopUML profila specifikācija, kas apvieno TFM matemātiskās topoloģijas formālismu ar OMG specifikāciju standartu, un izstrādātā programmatūras analīzes un projektēšanas metode. Sagatavotā modelēšanas metode nodrošina TopUML diagrammu lietošanu formālā programmatūras izstrādes procesā un sastāv no formalizētām projektēšanas aktivitātēm, ļaujot definēt risinājumu atbilstoši problēmvides funkcionēšanas īpašībām un viennozīmīgi izsekot cēloņseku attiecībām kā problēmvides, tā lietojumvides artefaktos. Pateicoties tam, ka TopUML valoda ir sagatavota kā UML valodas profils, tā var tikt ieviesta esošajos UML modelēšanas rīkos, kuri atbalsta profilus un to implementēšanu.

Darba rezultātu aprobācija

Par darba galvenajiem rezultātiem ziņots šādās **starptautiskās zinātniskajās konferencēs** (divas no konferencēm notika Latvijā, pārējās – citās valstīs):

1. *7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012)*, Vroclava, Polija, 2012. gada 29.-30. jūnijs,
2. *13th International Conference on Enterprise Information Systems (ICEIS 2011)*, Pekina, Ķīna, 2011. gada 8.-11. jūnijs,
3. *3rd International Workshop on Model-Driven Architecture and Modeling Driven Software Development (MDA & MDSD 2011) in conjunction with 6th International Working Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2011)*, Pekina, Ķīna, 2011. gada 8.-11. jūnijs,
4. *2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development (MDA & MTDD 2010) in conjunction with 5th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2010)*, Atēnas, Grieķija, 2010. gada 22.-24. jūlijs,
5. *13th East-European Conference on Advances in Databases and Information Systems (ADBIS 2009)*, Rīga, Latvija, 2009. gada 7.-10. septembris,
6. *4th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2009)*, Milāna, Itālija, 2009. gada 9.-10. maijs,
7. *Rīgas Tehniskās universitātes 49. starptautiskā zinātniskā konference*, Rīga, Latvija, 2008. gada 13.-15. oktobris.

Darba rezultāti publicēti šādos starptautiski recenzētos izdevumos:

1. Donins U. Semantics of Logical Relations in Topological Functioning Model// Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012) - 2012. (pieņemta publicēšanai)
2. Donins U., Osis J., Asnina E., Jansone A. Formal Analysis of Objects State Changes and Transitions// Proceedings of the 7th International Conference on

Evaluation of Novel Approaches to Software Engineering (ENASE 2012) - 2012. (pieņemta publicēšanai)

3. Donins U., Osis J. Topological Modeling for Enterprise Data Synchronization System: A Case Study of Topological Model-Driven Software Development// Proceedings of the 13th International Conference on Enterprise Information Systems, Volume 3. - Pekina, Ķīna: SciTePress, 2011. - 87.-96. lpp. [Datu bāzes: Thomson Reuters, Inspec, EI, DBLP, ISTP]
4. Donins U., Osis J., Slihte A., Asnina E., Gulbis B. Towards the Refinement of Topological Class Diagram as a Platform Independent Model// Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development. - Pekina, Ķīna: SciTePress, 2011. - 79.-88. lpp. [Datu bāzes: Thomson Reuters, Inspec, EI, DBLP]
5. Slihte A., Osis J., Donins U., Asnina, E., Gulbis, B. Advancements of the Topological Functioning Model for Model Driven Architecture Approach// Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development. - Pekina, Ķīna: SciTePress, 2011. - 91.-100. lpp. [Datu bāzes: Thomson Reuters, Inspec, EI, DBLP]
6. Asnina E., Gulbis B., Osis J., Alksnis G., Donins U., Slihte A. Backward Requirements Traceability within the Topology-based Model Driven Software Development// Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development. - Pekina, Ķīna: SciTePress, 2011. - 36.-45. lpp. [Datu bāzes: Thomson Reuters, Inspec, EI, DBLP]
7. Slihte A., Osis J., Donins U. Knowledge Integration for Domain Modeling// Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development. - Pekina, Ķīna: SciTePress, 2011. - 46.-56. lpp. [Datu bāzes: Thomson Reuters, Inspec, EI, DBLP]
8. Donins U. Software Development with the Emphasis on Topology// Advances in Databases and Information Systems (Lecture Notes in Computer Science, Vol.5968). - Berlīne, Vācija: Springer-Verlag, 2010. - 220.-228. lpp. [Datu bāzes: SCOPUS, Springer, DBLP]
9. Osis J., Donins U. Platform Independent model Development by Means of Topological Class Diagrams// Proceedings of the 2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development - Portugāle: SciTePress, 2010. - 13.-22. lpp. [Datu bāzes: SCOPUS, Thomson Reuters, Inspec, DBLP]
10. Osis J., Donins U. Formalization of the UML Class Diagrams// Evaluation of Novel Approaches to Software Engineering (Communications in Computer and Information Science (CCIS), Volume 69). - Berlīne, Vācija: Springer-Verlag, 2010. - 180.-192. lpp. [Datu bāzes: SCOPUS, Springer]

11. Osis J, Donins U. Modeling Formalization of MDA Software Development at the Very Beginning of Life Cycle// Advances in Databases and Information Systems. 13th East-European Conference, ADBIS 2009: Associated Workshops and Doctoral Consortium, Local Proceedings. - Rīga, Latvija: JUMI Publishing House Ltd., 2009. - 48.-61. lpp. [ISBN 978-9984-30-163-1]
12. Osis J., Donins U. An Innovative Model Driven Formalization of the Class Diagrams// Proceedings of the 4th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2009). - Portugāle: INSTICC Press, 2009. - 134.-145. lpp. [Datu bāzes: SCOPUS, Thomson Reuters, Inspec, DBLP]
13. Donins U., Osis J. Reconciling Software Requirements and Architectures within MDA// RTU zinātniskie raksti, Datorzinātne (sērija 5), Lietišķās datorsistēmas (sējums 38). - Rīga: RTU izdevniecība, 2009. - 84.-95. lpp. [Datu bāzes: DBLP]

Papildus rakstiem starptautiski recenzētos izdevumos, ir publicēts **metodiskais līdzeklis**:

1. Doniņš U. Topoloģiskā biznesa sistēmu modelēšana un programmatūras sistēmu projektēšana. - Rīga, Latvija: RTU izdevniecība, 2011. - 65 lpp. [ISBN: 978-9934-10-136-6]

Darba struktūra

Promocijas darbs satur ievadu, 5 nodaļas, secinājumus, literatūras sarakstu, 14 pielikumus, 71 zīmējumu un ilustrāciju, 32 tabulas, kopā 224 lappuses. Literatūras sarakstā ir 134 nosaukumi.

Ievadā dota pētījuma motivācija, promocijas darba mērķis un uzdevumi tā sasniegšanai, pētījuma zinātniskā novitāte un praktiskā vērtība, kā arī darba aprobācija un sasniegtie galvenie rezultāti.

Pirmajā nodaļā atspoguļota UML valodas analīze, kas parāda ieguvumus un trūkumus, lietojot šo valodu programmatūras projektēšanai un tai sekojošajai izstrādei. Lai novērstu atklātos trūkumus, ir identificētas UML valodas uzlabošanas iespējas. Papildus analīzē ietverts pārskats par tās attīstību.

Otrajā nodaļā analizētas programmatūras sistēmu projektēšanas un izstrādes metodes, kas analīzes un projektēšanas artefaktu izstrādei lieto UML valodu. Analīzes rezultāti parāda pozitīvos un negatīvos aspektus UML valodas virzītām modelēšanas metodēm.

UML paplašināšanas mehānisms – profilu izstrāde – ir apskatīta darba *trešajā nodaļā*. Šī nodaļa ietver TopUML profila izstrādi, kas paplašina UML valodas 2.4.1 versiju. TopUML profils apvieno UML valodu un TFM formālismu. UML valoda ir paplašināta, tai pievienojot TFM un papildinot UML diagrammas ar topoloģiskās funkcionēšanas īpašībām.

Ceturtā nodaļa ir veltīta TopUML modelēšanas metodes izstrādei, kas paredz sistemātisku TopUML profila lietošanu programmatūras izstrādes analīzes un projektēšanas fāzēs. TopUML metode ir definēta kā modelēšanas aktivitāšu kopums, kur katra aktivitāte definē tās ieejas un izejas artefaktus. Papildus ceturtā nodaļa ietver TopUML modelēšanas metodes salīdzinājumu ar otrajā nodaļā analizētajām metodēm.

Piektā nodaļa ietver TopUML profila un modelēšanas metodes aprobāciju eksperimentālas programmatūras sistēmas projektēšanas kontekstā, kā arī reālas programmatūras izstrādes kontekstā. Eksperimentālās programmatūras sistēmas projektēšana veikta arī sadarbībā ar divām ekspertu grupām, iegūtie rezultāti doti apkopotā veidā.

Secinājumos kopsavilkuma veidā apkopoti pētījuma gaitā iegūtie rezultāti, kā arī apskatīti iespējamie turpmāko pētījumu virzieni.

Promocijas darbs iekļauj šādus pielikumus: 1) Izmantotie saīsinājumi, 2) TopUML profila specifikācija, 3) Saistības starp TopUML profila diagrammām, 4) Veļas mazgātavas funkcionēšanas neformāls apraksts, 5) Veļas mazgātavas programmatūras sistēmas funkcionālās prasības un sistēmas mērķi, 6) Veļas mazgātavas sistēmas funkcionālās īpašības, 7) Veļas mazgātavas topoloģiskās telpas noslēgšanas operācijas izpilde, 8) Veļas mazgātavas uzvedības secību diagrammas, 9) Veļas mazgātavas topoloģiskā klašu diagramma, 10) SIA „Lattelecom Technology" apliecinājums TopUML metodes lietojumam programmatūras izstrādes projektā, 11) Organizācijas datu sinhronizācijas sistēmas specifikācija, 12) Organizācijas datu sinhronizācijas sistēmas funkcionālās īpašības, 13) Modelēšanas zināšanu pašnovērtējuma anketa, 14) „OMG-Certified UML Professional" sertifikāts.

1. VIENOTĀ MODELĒŠANAS VALODA – STANDARTS PROGRAMMATŪRAS PROJEKTĒJUMA SPECIFICĒŠANAI

UML valoda ir modelēšanas notācija, kas ir paredzēta sistēmu artefaktu[1] vizualizēšanai, specificēšanai, konstruēšanai un dokumentēšanai [57]. Tā ietver standarta līdzekļus kā konceptuālu, tā konkrētu lietu aprakstīšanai [28]. Drīz pēc UML valodas pirmās versijas apstiprināšanas un publicēšanas 1997. gadā tā kļuva par standartu objektorientētas sistēmanalīzes un projektēšanas veikšanai [42] un joprojām tāda ir šodien [14]. Līdz ar pirmās UML valodas specifikācijas publicēšanu vairākas pētnieku un praktiķu grupas ir veikušas pētījumu par tās lietošanu programmatūras izstrādē un uzlabošanas iespējām, iegūtos rezultātus atspoguļojot zinātniskajās publikācijās un grāmatās. UML valodas attīstības aktuālo pētījumu virzieni:

UML valodas semantikas formalizācija ([23], [34] (pēc 1.1 versijas publicēšanas), [89] (pēc 2.0 versijas publicēšanas)),

UML valodas paplašināšana ([49], [69], kā arī pārskats par dažādu pētnieku grupu sagatavotajiem UML valodas profiliem [75]),

UML diagrammu izstrādes formalizēšana ([64] un [67]),

UML valodas modelēšanas elementu ontoloģiskā analīze ([92]),

Empīriskie UML valodas lietošanas novērtējumi ([14] un [24]),

UML valodas sarežģītības analīze ([22], [81] un [82]),

Grūtības mācoties UML valodu ([83]) un kā tās novērst ([7]),

Transformācijas starp UML diagrammām ([46], [43] un [50]),

[1] Artefakts programmatūras izstrādē ir vienība, kas ir radīta vai saņemta izstrādes procesa laikā, piemēram, lietošanas gadījumi, prasības, projektējums, pirmkods, izpildāmie faili.

Programmatūras koda ģenerēšana un saistītās problēmas ar ģenerētā koda kvalitāti ([45] un [80]),

Eksperimenti UML modeļu efektivitātes aspektu novērtēšanai ([12]).

Pētījumu skaits UML valodas attīstībai un stiprināšanai ir cieši saistīts ar šīs valodas pirmsākumiem. Dobings (*Dobing*) un Pārsons (*Parson*) [14] uzsver, ka „*UML valoda netika sagatavota pamatojoties uz teorētiskiem principiem, kas nepieciešami efektīvai un lietojamai analīzes un projektēšanas metodei. Tā vietā UML valoda tika radīta apvienojot vairākas, atsevišķos jautājumos pat konfliktējošas, „labākās prakses" no dažādām programmatūras inženierijas daļām (piemēram, Booch metodi* [8]*, Objektu modelēšanas tehniku* [77]*, Objektorientēto programmatūras inženieriju* [36])".

Šajā nodaļā veiktā UML valodas analīze ir balstīta uz tās 2.4.1 versijas specifikāciju, kas ir sadalīta divās daļās: infrastruktūra [56] (definē pamata metamodeli) un superstruktūra [57] (definē diagrammas, to elementu notāciju un semantiku). Jāatzīmē, ka superstruktūras specifikācija ir balstīta uz infrastruktūras specifikāciju. Modelēšanas koncepti UML valodā ir sadalīti horizontālos slāņos, attēlojot attiecīgu saderības līmeni (saderības līmenis ir jāņem vērā izstrādājot vai izvēloties modelēšanas rīku [28]).

1.1. Vienotās modelēšanas valodas formālisms un tā paaugstināšana

UML valodas specifikācija ir izstrādāta, lietojot metamodelēšanas pieeju, kas adaptē formālas specificēšanas tehnikas. Metamodelis tiek izmantots, lai specificētu UML valodu veidojošo modeli. Neskatoties uz metamodelēšanas pieejas lietošanu, UML valodas specifikāciju nevar uzskatīt par formālu valodas specifikāciju dēļ dabīgās valodas (angļu valodas) izmantošanas semantikas definēšanā. UML valodas specifikācijā [56] ir uzsvērts, ka metamodelēšanas lietošana neliedz formalizēt to, izmantojot kādu no formālajām vai matemātiskajām valodām, kā, piemēram, OCL [91], Z [87], PVS [74] vai RAISE [52]. Papildus formalizācijas ieviešanai UML valodas specifikācijā ir šādi ieguvumi [23]: skaidrība, saskanība, paplašināmība, bagātināšana, pierādīšana un modelēšanas rīki, kuru lietošana prasa precīzu UML valodas semantiku.

Daļa no UML valodas formalizēšanas pētījumiem ir veltīti tikai modeļu semantikai, kamēr citi pētījumi – spriešanai par modeļiem un to savstarpējām transformācijām. Formālas UML semantikas specificēšanai ir sagatavotas vairākas pieejas, kas paredz dažādu paņēmienu lietošanu, piemēram, formālās valodas Z ([23]) vai Object-Z ([40]), kategoriju teoriju ([1] un [13]), plūsmas teoriju ([11]), π-rēķinus jeb procesu algebru ([93]) vai matemātiskas izteiksmes ([89]). UML valodas semantikas formalizācijas pētījumi ir paredzēti tās iekšējās saskanības uzlabošanai, nevis tās elementu saistību definēšanai ar problēmvidēm [25]. Lai sasaistītu UML valodas elementus ar problēmvides elementiem, tiek veikti pētījumi programmatūras analīzes un projektēšanas procesa formalizēšanai ([16], [65]), kā arī UML valodas elementu aprakstīšanai, lietojot ontoloģiju ([25], [92]).

1.2. Ieguvumi un trūkumi vienotās modelēšanas valodas lietošanā

UML valodas lietošanai programmatūras izstrādes procesā ir gan ieguvumi, gan trūkumi. Lielākie ieguvumi ir šādi ([2], [14], [27], [57], [59]): UML valoda ir neatkarīga no programmatūras izstrādes metodēm, tehnikām un platformām; tā satur paplašināšanas mehānismus, ļaujot risināt specifiskus uzdevumus; izstrādātie modeļi ir savietojami starp dažādu ražotāju piedāvātajiem modelēšanas rīkiem, pateicoties metadatu apmaiņas (*XML Metadata Interchange*, XMI) standarta izmantošanai. Lielākie trūkumi ir šādi ([14], [16], [23], [39], [69], [84]): izmērs, nesaskanība, neviennozīmīgums, valodas apakškopas un cēloņseku sakarību nepietiekamība. No šiem trūkumiem izriet vairākas problēmas – neskaidra semantika, nepareizi norādījumi izstrādes procesā, nekorekta problēmvides funkcionēšanas analīze, adekvātu modelēšanas rīku trūkums, nespēja trasēt cēloņseku attiecības starp problēmvidē eksistējošiem artefaktiem un sagatavotajiem lietojumvides artefaktiem.

1.3. Kopsavilkums

Aplūkojot tuvāk ieguvumus un trūkumus UML valodas lietošanā programmatūras izstrādes procesā ir redzams, ka daļa no ieguvumiem kļūst par trūkumiem, piemēram, neatkarība no programmatūras izstrādes metodēm noved pie nepareiziem norādījumiem izstrādes procesā. Lai novērstu identificētās UML valodas un tās lietošanas nepilnības, tiek veikti pētījumi UML valodas stiprināšanai un formalizēšanai.

Viens no UML valodas stiprināšanas paņēmieniem ir matemātiskās topoloģijas lietošana, tādejādi novēršot cēloņseku nepietiekamību [16]. Šādā gadījumā UML valoda tiek papildināta ar TFM topoloģiskajām un sistēmu funkcionēšanas īpašībām, izveidojot jaunu UML valodu saimē ietilpstošu modelēšanas valodu – *Topoloģisko vienoto modelēšanas valodu* (saīsināti *TopUML*). TopUML profila pamats ir piedāvāts un publicēts [69]. Pirmie pētījumu rezultāti TopUML izveidē parāda, ka topoloģisko un sistēmu funkcionēšanas īpašību pārnešana no TFM uz UML valodu ļauj skaidri definēt un trasēt cēloņseku attiecības starp problēmvidē eksistējošiem artefaktiem un sagatavotajiem lietojumvides artefaktiem.

Nākamā nodaļa ir veltīta UML valodas vadīto modelēšanas metožu analīzei, lai novērtētu iespēju ar modelēšanas metožu palīdzību novērst pārējās identificētās UML valodas un tās lietošanas nepilnības.

2. PROGRAMMATŪRAS PROJEKTĒŠANA, LIETOJOT UML VADĪTĀS MODELĒŠANAS METODES

UML valoda ir notācija un līdz ar to tās specifikācija neietver vadlīnijas tās lietošanai programmatūras izstrādes procesā. Neskatoties uz to, ka UML valoda ir neatkarīga no modelēšanas metodēm, lielākā daļa no šīm metodēm izmanto lietošanas gadījumu vadītu pieeju [14]. Šo faktu var izskaidrot ar UML valodas autoru Būča (*Booch*), Rambo (*Rumbaugh*) un Jākobsona (*Jacobson*) ietekmi, jo savā grāmatā „*Vienotās modelēšanas valodas lietotāja ceļvedis*" („*The Unified Modeling Language User Guide*", [10]) viņi iesaka izmantot tieši lietošanas gadījumu vadītu procesu. Kopš šī lietotāja ceļveža publicēšanas liela

daļa no UML valodas vadītajām modelēšanas metodēm attīsta lietošanas gadījumu vadītu modelēšanu (piemēram, [44], [76] un [88]). Dažādās metodēs lietošanas gadījumu tekstuālie apraksti tiek veidoti atšķirīgi un savādākā formātā – UML valodas specifikācijā nav ietvertas vadlīnijas šo aprakstu izstrādei un struktūrai. UML valodas specifikācijā [57] ir teikts, ka „lietošanas gadījumi tipiski tiek specificēti dažādos formātos, kā, piemēram, dabīgā valoda, tabulas, kokveida grafi. Tāpēc lietošanas gadījumu apraksta struktūru nav iespējams noteikt nedz precīzi, nedz vispārīgi ar formāla modeļa palīdzību."

Veiksmīgs programmatūras izstrādes projekts var tikt mērīts ar tādiem kritērijiem, kā: veiktās piegādes, piegāžu grafiks, sagatavotā rezultāta elastību attiecībā pret izmaiņām un tā spēju adaptēties. Lai programmatūras projekts būtu uzskatāms par veiksmīgu attiecībā pret minētajiem kritērijiem, tam ir jāatbilst šādam raksturojumam [9]:

1. programmatūras risinājumam ir jābūt izteiktai arhitektūras vīzijai,
2. jāizmanto pārvaldāms programmatūras izstrādes dzīvescikls.

Programmatūras arhitektūra ir „fundamentāla sistēmas organizācija, kas ietverta tās komponentos, komponentu savstarpējās attiecībās un attiecībās ar apkārtējo vidi, un pamatprincipu lietošana projektēšanas un attīstības nodrošināšanai" [35]. Ieteicamo programmatūra arhitektūru raksturo šādas pazīmes [9]:

1. tā ir izstrādāta nodalītos abstrakcijas slāņos,
2. tai ir skaidri nodalītas atbildības starp katra slāņa saskarni un implementāciju,
3. arhitektūra ir vienkārša – funkcionalitāte tiek panākta ar kopēju abstrakciju un mehānismu lietošanu.

Pašlaik eksistē vairākas UML valodas vadītās modelēšanas metodes, kas paredzētas programmatūras projektēšanai un izstrādei, piemēram, programmatūras izstrādes dzīvescikli [79], lietošanas gadījumu vadītas metodes [76], modeļvadāmā arhitektūra [41], paraugos balstīta izstrāde [44], komponentu balstīta izstrāde [88] un konceptuālā modelēšana [53]. Darbā veiktais programmatūras izstrādes metožu pārskats ietver pašreiz esošo UML valodas vadīto modelēšanas metožu analīzi, lielāko uzsvaru un uzmanību veltot UML valodas diagrammu lietošanai un izstrādei – kādiem nolūkiem tās tiek izmantotas un to izstrādes secībai. Veiktā analīze papildus parāda, vai apskatītā metode ietver transformācijas likumus vai vadlīnijas starp dažādu veidu diagrammām. Veiktais pārskats ietver tādas metodes, kas tiek izmantotas programmatūras izstrādes industrijā [14], formalizē izstrādes procesu un problēmvidi [64] un kas tiek izmantotas konjukcijā ar programmatūras izstrādes rīkiem [47].

2.1. Ieguvumi un trūkumi UML valodas vadīto metožu lietošanā

Apvienojot UML valodu ar kādu no modelēšanas metodēm tiek iegūts spēcīgs mehānisms kā problēmvides, tā programmatūras sistēmas izprašanai, kā arī plānotās programmatūras sistēmas projektēšanai. Neskatoties uz faktu, ka UML valodas vadītās modelēšanas metodes nodrošina sistemātisku UML diagrammu lietošanu, šīs metodes pārklāj dažādas programmatūras izstrādes dzīves cikla daļas, attiecīgi izmantojot tikai daļu no UML diagrammām. Šī iemesla dēļ programmatūras izstrādātājiem ir jāapvieno vairākas modelēšanas metodes, tādejādi UML valodas lietošana tiek padarīta vēl sarežģītāka un

neizprotamāka. Visu programmatūras izstrādes dzīves ciklu aptver tikai Vienotais process (*Unified process*) [3] un Microsoft risinājumu ietvars (*Microsoft Solutions Framework*) [90], kamēr citas metodes fokusējas vairāk uz analīzes fāzi (piemēram, Biznesa objektorientētā modelēšana (B.O.O.M.) [76], TFM modeļvadāmajai arhitektūrai (TFMfMDA) [4] un konceptuālā modelēšana [59]) vai vairāk uz projektēšanas fāzi un mazāk uz analīzes fāzi (piemēram, šablonu [44] un komponentu balstīta izstrāde [88]). Apkopojot UML diagrammu lietošanu modelēšanas metodēs, ir redzams, ka ne visi diagrammu veidi tiek izmantoti (šādā veidā modelēšanas metodes nosaka un ietekmē izmantoto UML diagrammu apjomu praksē). Starp analizētajām metodēm vienotajā procesā tiek lietots visvairāk diagrammu veidu.

Vienotā procesa lietošanas ieguvums ir pilna programmatūras izstrādes dzīves cikla pārklājums, savukārt par tā trūkumu kļūst lietošanas gadījumu vadītā problēmvides analīze. Ar lietošanas gadījumiem vienotajā procesā nav iespējams nodrošināt formālu problēmvides funkcionēšanas analīzi un formalizāciju. Starp visām apskatītajām metodēm tikai viena – TFMfMDA – piedāvā formālu problēmvides funkcionēšanas analīzi. Tā izmanto TFM kā tehniku gan problēmvides, gan lietojumvides analīzei un formalizēšanai. Ieguvums no TFMfMDA metodes lietošanas ir programmatūras dzīves cikla sākuma formalizēšana, bet trūkums ir konceptuālās klašu diagrammas lietošana un izstrāde. TFM apraksta kā problēmvides, tā lietojumvides funkcionēšanu, ietverot atbildības visas sistēmas ietvaros. Transformējot TFM uz konceptuālo klašu diagrammu, šī nepieciešamā informācija par objektiem veicamajiem uzdevumiem netiek pārnesta. Rezultātā tiek iegūtas klases bez to atbildībām. „*Izlemt, kuras operācijas pieder kuram objektam, un kā tiem savstarpēji jāsadarbojas, it ļoti svarīgi un sarežģīti. Šis ir kritisks solis, kas atrodas centrā tam, ko nozīmē izstrādāt objektorientētu sistēmu, nevis domēna modeļu un citu diagrammu zīmēšana*" [44].

UML valodas lietošanas analīze programmatūras industrijā uzrāda, ka piecas visvairāk izmantotās diagrammas ir klašu, lietošanas gadījumu, secību, aktivitāšu un stāvokļu diagramma [14]. Šie rezultāti ir cieši saistīti ar UML valodas vadītām modelēšanas metodēm – darba ietvaros veiktais metožu apskats uzrāda, ka arī starp tām pieci visbiežāk lietotie diagrammu veidi ir tie paši, kas minēti [14].

2.2. Kopsavilkums

Modelēšanas metožu lietošana programmatūras izstrādes procesā samazina un pat novērš dažus no UML valodas trūkumiem, kas uzskaitīti darba pirmajā nodaļā. Samazinātie un novērstie trūkumi ir:

- *izmērs* – sistemātiskas un secīgas programmatūras analīzes un projektēšanas aktivitātes risina problēmas saistītas ar lielo UML valodas elementu un diagrammu skaitu;
- *nesaskanība* – modelēšanas metodes, izmantojot iepriekš definētas aktivitātes, paredz diagrammas izstrādāt vienu pēc otras, tādejādi parādot saistības un pārejas starp diagrammām;

- *neviennozīmīgums* – UML valodas semantika kopā ar metodisku UML diagrammu konstruēšanu veido vienotu sapratni starp ieinteresētajām pusēm;
- *valodas apakškopas* – piedāvājot UML valodas paplašinājumu (piemēram, profilu) kopā ar pietiekamu tā lietošanas metodi vai aprakstu tiek uzskatāmi parādītas paplašinājuma attiecības ar esošajiem UML valodas elementiem, kā arī ieguvums programmatūras izstrādes procesā no izstrādātā paplašinājuma.

Diemžēl daļējais programmatūras dzīvescikla pārklājums un fragmentārais UML diagrammu lietojums analizētajās modelēšanas metodēs augstāk uzskaitītos trūkumus nenovērš pietiekamā apjomā, kāds nepieciešams efektīvai un lietojamai analīzes un projektēšanas metodei. Efektīva un lietojama metode ir tāda, kas ļauj sasniegt sagaidāmo rezultātu (problēmvides formalizēšanu un lietojumvides projektēšanu atbilstoši tai identificētajām funkcionēšanas īpašībām) un kas piedāvā pietiekamus līdzekļus viennozīmīgai cēloņseku identificēšanai kā problēmvides, tā lietojumvides artefaktos.

3. VIENOTĀS MODELĒŠANAS VALODAS UZLABOŠANA

UML valoda ir paplašināma divos veidos – lietojot tās profilu definēšanas mehānismu vai MOF (*Meta Object Facility* [54]) balstītu metamodeļu definēšanu [56]. Izstrādājot profilu tiek iegūts jauns UML valodas dialekts (profilā ir pieejami visi UML valodas elementi kopā ar jaunajiem tajā definētajiem), savukārt MOF balstītā pieeja paredz esošo metamodeļu pārdefinēšanu un jaunu metamodeļu specificēšanu, atbilstoši metamodelēšanas principiem. Jāatzīmē, ka lietojot MOF balstīto pieeju tiek zaudēti visi ieguvumi no profila izstrādes. Pirms jauna UML valodas paplašinājuma izstrādes ir nepieciešams noteikt nepieciešamās modelēšanas valodas apjomu:

- ja tiek izmantota lielākā daļa no UML valodas elementiem, tad piemērotākais risinājums ir jauna profila izstrāde,
- ja paredzēts izmantot tikai nelielu daļu no UML valodas elementiem vai ir nepieciešams lietot sarežģītākas UML valodas iespējas (piemēram, pārdefinēšanu), tad piemērotākais risinājums ir lietot MOF balstītu paplašinājuma definēšanas pieeju.

Raugoties no lietojamības un praktiskuma viedokļa, piemērotākais UML valodas uzlabošanas paņēmiens ir jauna profila izstrādāšana. Viens no ieguvumiem šādai uzlabošanas pieejai ir iespēja jauno profilu lietot jau esošajos modelēšanas rīkos, kuri atbalsta UML valodu un tās profilu implementēšanu [78]. Šādā veidā tiek samazinātas jaunās modelēšanas valodas ieviešanas izmaksas un adaptēšanas ilgums programmatūras izstrādes uzņēmumos.

3.1. Vienotās modelēšanas valodas profila izstrādāšana

UML valodas profila izstrāde ir jāveic sistemātiskā veidā, lietojot vienotu profilu specificēšanas pieeju vai šablonu. Tā kā UML valodas specifikācija satur tikai profilu definēšanas elementus, bet neietver vadlīnijas jaunu profilu specificēšanai, pirms profila izstrādes ir jāveic tā specificēšanas vadlīniju sagatavošana. Darba ietvaros ir apskatīti četri

dažādi UML valodas profili (*Executable UML* (saīsināti *xUML*) [49], *TFMfMDA* [4], *Object Modeling Group System Modeling Language* (saīsināti *OMG SysML*) [58] un *Service Oriented Architecture Modeling Language* (saīsināti *SoaML*) [55]), lai noteiktu pēc iespējas labāku šablonu jauna profila specificēšanai. Veiktā profilu analīze parāda, ka to definēšanai nav izmantota vienota specificēšanas metode – katrs no profiliem ir definēts pēc to autoru ieskatiem (iegūtais secinājums sakrīt ar [75] atspoguļoto aptuveni 50 profilu analīzes rezultātu). Tikai OMG SysML un SoaML profilu specifikāciju struktūrā ir vērojamas diezgan lielas līdzības – aptuveni 85% no specifikācijas struktūras ir vienāda. Šo divu profilu specifikācijas struktūra ir veidota un strukturēta pēc iespējas tuvinot tās UML valodas specifikācijai. Šādā veidā specificējot UML profilu tiek uzlabota tā lasāmība un saprotamība. Pamatojoties uz izdarīto secinājumu darba ietvaros ir definēts šablons UML profila specifikācijai, kura izstrāde apskatīta nākamajā apakšnodaļā.

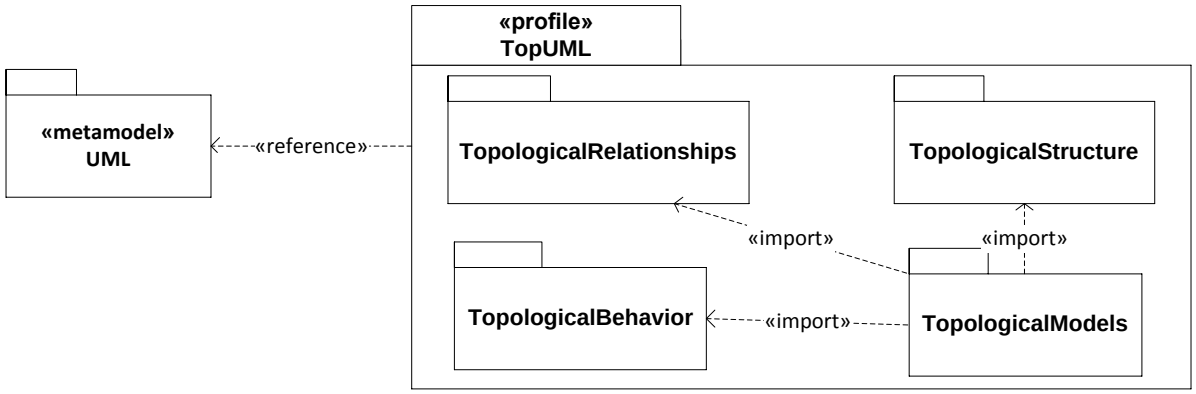
3.2. Topoloģiskā vienotā modelēšanas valoda

Galvenais UML valodas uzlabošanas mērķis ir bagātināt tās elementus ar TFM ietilpstošo matemātisko topoloģiju un formālismu, tādejādi novēršot cēloņseku sakarību trūkumu esošajā valodas specifikācijā. Topoloģiskā vienotā modelēšanas valoda (TopUML) apvieno UML valodu ar TFM formālismu un ir balstīta uz MOF metamodelēšanas principiem. Ideja par TopUML valodu ir adaptēta no [60], kur ir parādīts matemātiskā formālisma trūkums UML diagrammu izstrādes laikā.

TopUML ir izstrādāta kā UML profils, tā specifikācija lieto UML valodas pakotņu apvienošanas iespējas, lai integrētu nepieciešamos papildinājumus UML metamodelī. TopUML profila izstrāde ir veikta, izpildot divus uzdevumus:

1. papildināt UML valodu, lietojot tās paplašināšanas – profilu definēšanas – mehānismu, tādejādi radot TopUML profilu,
2. definēt formalizētu modelēšanas metodi TopUML profila lietošanai problēmvides analīzei un tai atbilstošas lietojumvides projektēšanai programmatūras izstrādē.

Izstrādātais TopUML profils ietver visas UML valodas 2.4.1 versijā definētās četrpadsmit diagrammas, kā arī vienu jaunu diagrammu – Topoloģisko funkcionēšanas modeli. Veiktā esošo UML diagrammu analīze parāda, ka papildus TFM definēšanai ir nepieciešams papildus bagātināt divas diagrammas: klašu un lietošanas gadījumu, lai sasniegtu izvirzīto TopUML profila definēšanas mērķi – cēloņseku attiecību izsekošanai kā problēmvides elementos, tā lietojumvides elementos. Paplašinātās UML valodas diagrammas tiek sauktas attiecīgi „*Topoloģiskā klašu diagramma*" un „*Topoloģiskā lietošanas gadījumu diagramma*". TopUML profils sastāv no četrām pakotnēm, astoņiem stereotipiem, diviem uzskaitījumiem un trīs metamodeļiem (viens metamodelis TFM un katrai no paplašinātajām diagrammām). Augstākā līmeņa TopUML profila diagramma atbilstoši UML valodas notācijai ir dota 3.1. attēlā, kas attēlo paplašināto metamodeli, profilu, profilā ietilpstošās pakotnes un to savstarpējās attiecības. Profila diagramma ir izstrādāta atbilstoši UML valodas specifikācijai, lietojot UML valodas elementus.



3.1. att. Augstākā līmeņa TopUML profila diagramma

Pakotnes TopUML profilā tiek lietotas, lai sagrupētu elementus atbilstoši to nozīmei un semantikai, kā arī lai atvieglotu turpmāku TopUML valodas attīstību – jaunu TopUML versiju izstrādāšanu. TopUML profilā ietilpst šādas pakotnes un stereotipi tajās:

- *TopologicalRelationships* – ietver elementus attiecību definēšanai:
  - *TopologicalRelationship* – specificē orientētu, bināru topoloģisko attiecību, kas ļauj definēt cēloņu un seku sakarības starp diviem elementiem,
  - *LogicalRelationsip* – specificē loģisko attiecību, kas ļauj definēt loģiskās sakarības (konjunkciju, disjunkciju un izslēdzošo disjunkciju) starp divām vai vairākām topoloģiskajām attiecībām,
- *TopologicalBehavior* – ietver elementus uzvedības definēšanai:
  - *FunctionalFeature* – specificē funkcionālo īpašību, kas atbilst nedalāmas biznesa darbības aprakstam; katra funkcionālā īpašība ir unikāls kortežs (stereotips *FunctionalFeature* ir šī korteža abstrakcija),
  - *Condition* – specificē sistēmās ietilpstošos nosacījumus (eksistē divu veidu nosacījumi: priekšnosacījums un pēcnosacījums); lai uzvedības elements varētu uzsākt darbību, visiem tā priekšnosacījumiem ir jābūt patiesiem; lai uzvedības elements varētu beigt savu darbību, visiem tā pēcnosacījumiem ir jābūt patiesiem,
  - *ActionResult* – specificē objekta darbības rezultātu un darbības laikā izmantotos objektus,
- *TopologicalStructure* – ietver elementus struktūras definēšanai:
  - *TopologicalCycle* – specificē orientētu sistēmas funkcionēšanas ciklu,
  - *TopologicalOperation* – uzvedības elements, kas specificē nosaukumu, tipu, parametrus un ierobežojumus, lai ierosinātu saistīto uzvedības elementu, un kas specificē saistītās funkcionālās īpašības un topoloģiskās attiecības cēloņseku sakarību definēšanai sistēmā,
- *TopologicalModels* – ietver UML valodai pievienoto diagrammas veidu:
  - *TopologicalFunctioningModel* – specificē TFM ar UML metamodelēšanas līdzekļiem. TFM ir matemātisks modelis, kas parāda sistēmu funkcionēšanu orientēta grafa veidā (grafa virsotnes ir funkcionālās īpašības, bet loki starp tām – topoloģiskās attiecības). Funkcionālās īpašības ietver sistēmas funkcionēšanas un

struktūras informāciju, savukārt topoloģiskās attiecības – cēloņseku sakarības starp tām.

TopUML profila pakotnes ir definētas tā, lai ietvertu visus nepieciešamos elementus TFM, Topoloģiskās klašu un Topoloģiskās lietošanas gadījumu diagrammas izstrādei. Visi definētie stereotipi tiek izmantoti vairākos diagrammu veidos, tādejādi samazinot specificējamo valodas elementu skaitu un novēršot dublējošu elementu esamību.

Jāatzīmē tas, ka novēršot UML valodai identificēto nepilnību – cēloņseku attiecību trūkumu – ar matemātiskās topoloģijas ieviešanu tās elementos un izstrādājot paplašinājumu TopUML profilu, izmēra un valodas apakškopu nepilnību radītais efekts tiek tikai palielināts. Lai samazinātu šo nepilnību ietekmi UML valodas paplašinājuma lietošanā, tam ir jāsagatavo arī atbilstoša modelēšanas metode. TopUML modelēšanas metodei ir jāietver šādi aspekti:

1. pienācīga problēmvides un lietojumvides analīze (visiem lietojumvides artefaktiem jābūt labi analizētu un izprastu problēmvides elementu abstrakcijām);
2. jāiekļauj pēc iespējas lielāks skaits diagrammu un jāpārklāj pēc iespējas lielāka daļa no programmatūras dzīves cikla;
3. sagatavotajiem artefaktiem jāveic tikai tiem paredzētie uzdevumi;
4. izstrādātajiem komponentiem jābūt mazā sasaistē ar pārējo sistēmu un ar labi definētu saskarni.

3.3. Kopsavilkums

UML profila specificēšana ir tikpat izaicinoša aktivitāte, kā citu UML diagrammu izstrāde, jo UML specifikācija definē tikai modelēšanas valodu ar visiem tās elementiem. Savukārt diagrammu izstrādāšanas vadlīnijas un norādījumi šādā notācijas specifikācijā nav ietverti. Līdz ar to pirms jauna UML profila izstrādes ir nepieciešams noteikt tā specificēšanas metodi un struktūru. Gan darba ietvaros veiktā UML profilu analīze, gan sistemātisks publicēto profilu apskats [75] norāda uz šādas metodes un struktūras trūkumu – autori UML profilus definē pēc saviem ieskatiem. Līdz ar to tiek apgrūtināta šo profilu saprotamība un apgūšana. Ņemot vērā analizēto profilu specifikācijas struktūru, darbā ir definēts UML profila specificēšanas šablons, kas ir izmantots TopUML profila definēšanai. Ieteicamā profilu specifikācijas struktūra ir pietuvināta un daļēji atbilst UML valodas specifikācijai.

Pieņemot lēmumu par jaunas modelēšanas valodas izstrādi, vispirms ir jānosaka tās apjoms un cik lielā mērā tiks izmantota jau esoša valoda. Ja jaunajā modelēšanas valodā ir paredzēts lietot lielāko daļu no UML valodas elementiem, tad ieteicams ir izstrādāt profilu (šādā gadījumā ir pieejami visi UML valodas elementi, kā arī jaunie, profilā definētie, elementi). Savukārt, ja paredzēts lietot tikai nelielu daļu no UML valodas, tad piemērotāks scenārijs ir MOF balstīta metamodelēšana. Papildus ieguvums no profilu lietošanas ir iespēja to implementēt kādā no esošajiem UML modelēšanas rīkiem.

Darba ietvaros specificētais TopUML profils papildina UML valodu ar TFM topoloģiskajām un funkcionēšanas īpašībām, tādejādi novēršot cēloņseku sakarību trūkumu esošajā valodas specifikācijā. Izstrādātais profils nodrošina elementus, kas nepieciešami cēloņseku sakarību definēšanai kā problēmvidē, tā lietojumvidē. Papildus valodas elementu

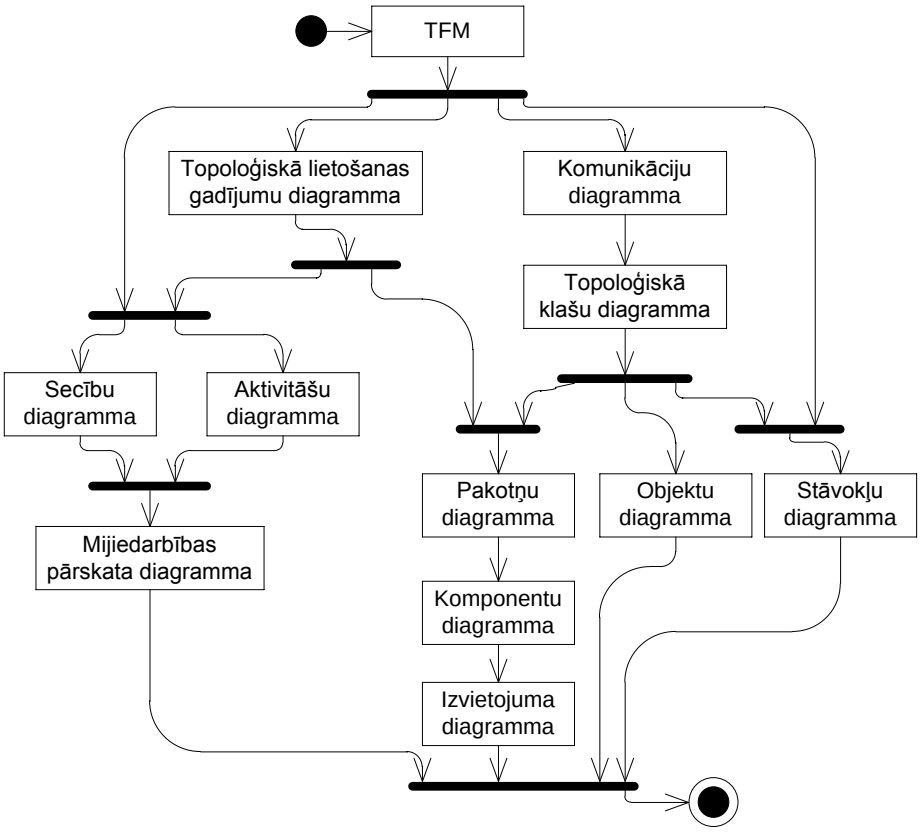
definīcijām ir dotas sakarības starp šiem elementiem, lai nodrošinātu transformāciju iespējamību starp izstrādātajām diagrammām. Lai šos valodas elementus izmantotu pietiekamai cēloņseku sakarību definēšanai, ir nepieciešams lietot atbilstošu modelēšanas metodi. Nākamajā darba nodaļā tiek definēta šāda metode – TopUML modelēšanas metode.

4. TOPUML MODELĒŠANA – PROGRAMMATŪRAS PROJEKTĒŠANAS METODE

TopUML modelēšanas metode ir modeļvadāma problēmvides analīzes un lietojumvides projektēšanas pieeja, kas apvieno TFM un tā formālismu ar TopUML profila elementiem un diagrammām. Šī metode ir izstrādāta ar mērķi novērst UML valodā un tās lietošanas metodēs identificētos trūkumus, piemēram, cēloņseku sakarību ignorēšana, neviennozīmīgums un nesaskanība. Modeļvadāmās arhitektūras kontekstā TFM apskata problēmvides funkcionēšanu nošķirti no lietojumvides funkcionēšanas un pilnībā atspoguļo sistēmas funkcionalitāti no skaitļošanas neatkarīgā veidā, savukārt pārējās TopUML diagrammas satur elementus sistēmas projektējuma atspoguļošanai no platformneatkarīgā un platformspecifiskā skatu punkta.

4.1. Pārejas starp TopUML diagrammām

Pārejas starp diagrammām TopUML modelēšanas metodes ietvaros ir dotas 4.1. attēlā Aktivitāšu diagrammas veidā, kur diagrammas ir attēlotas kā objekti, bet saites starp tām kā objektu plūsma.



4.1. att. Pārejas starp TopUML diagrammām

Lielākā daļa no dotajām pārejām starp diagrammām ir automatizējama, bet izstrādāto modeļu validāciju ir jāveic problēmvides un lietojumvides ekspertam. Sākotnējā modeļa – TFM – izstrādi var daļēji automatizēt, kā parādīts [72], kur biznesa lietošanas gadījumu specifikācijas tiek transformētas par funkcionālajām īpašībām un topoloģiskajām attiecībām starp tām. Savukārt citas diagrammas tiek iegūtas transformējot un lietojot sagatavoto TFM (vēlākajās izstrādes aktivitātēs arī citu veidu diagrammas).

Diagrammas, kas tiek izstrādātas specificējot programmatūru ar TopUML modelēšanas metodi, ir dotas tabulā 4.1, kur ir parādīta diagrammu izstrādes secība (kolonnā „I.s.") un ar transformāciju palīdzību iegūstamās diagrammas. Uzskaitītā diagrammu izstrādes secība ir paredzēta „no augšas uz leju" specificēšanai.

Tabula 4.1

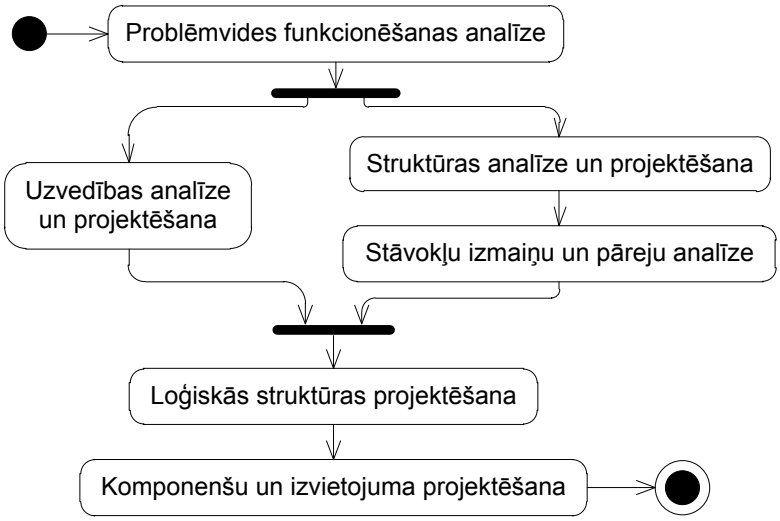
TopUML modelēšanas metodē ietvertās diagrammas un to izstrādes secība

Nr	TopUML diagramma	I. s.	Transformē-jama par	Apraksts
1.	Topoloģiskais funkcionēšanas modelis	1	Topoloģiskā lietošanas gadījumu, Secību, Aktivitāšu, Komunikāciju un Stāvokļu diagrammas	Sākotnējais TFM tiek izstrādāts analizējot problēmvides funkcionēšanu un tās nosacījumus. TFM bagātināšana paredz tā sākotnējās versijas modificēšanu atbilstoši izvirzītajām funkcionālajām prasībām un vēlamajām lietojumvides funkcionēšanas īpašībām. Bagātinot TFM tiek pārbaudītas funkcionālās prasības (atrastas konfliktējošas, trūkstošas un nepilnīgas prasības), kā arī pārbaudīts izstrādātais TFM (vai tas ietver visu lietojumvidei nepieciešamo funkcionēšanas informāciju).
2.	Topoloģiskā lietošanas gadījumu diagramma	2	Secību, Aktivitāšu un Pakotņu diagrammas	Lietošanas gadījumu apjoms tiek noteikts ar funkcionālo prasību vai sistēmas mērķu palīdzību. Katrā lietošanas gadījumā ietilpstošā funkcionalitāte tiek iegūta no TFM, atbilstoši noteiktajām saistībām starp tā funkcionālajām īpašībām un funkcionālajām prasībām vai sistēmas mērķiem.
3.	Secību diagramma	3	Mijiedarbības pārskata diagramma	Secību diagramma parāda ziņojumu sūtīšanu starp aktieriem un objektiem. Parasti tiek sagatavots komplekts ar Secību diagrammām – viena diagramma katram lietošanas gadījumam. Lietošanas gadījumi nosaka katrā Secību diagrammā atspoguļojamo informācijas apjomu, bet TFM nosaka sūtāmos ziņojumus un to sūtītājus, saņēmējus un secību.
4.	Aktivitāšu diagramma	3	Mijiedarbības pārskata diagramma	Aktivitāšu diagramma specificē lietošanas gadījumā ietverto darbplūsmu. Parasti tiek sagatavots komplekts ar Aktivitāšu diagrammām – viena diagramma katram lietošanas gadījumam. Lietošanas gadījumi nosaka katrā Aktivitāšu diagrammā atspoguļojamo informācijas apjomu, bet TFM nosaka darbības

Nr	TopUML diagramma	I. s.	Transformē-jama par	Apraksts
				un saites starp tām.
5.	Mijiedarbības pārskata diagramma	4	-	Definē mijiedarbības pārskatu, lietojot Aktivitāšu diagrammas paveidu. Šī diagramma lielāku uzsvaru liek uz vadības plūsmas pārskatu.
6.	Komunikāciju diagramma	2	Topoloģiskā klašu diagramma	Komunikāciju diagramma TopUML modelēšanas metodē tiek izmantota kā pārejas modelis starp TFM un Topoloģisko klašu diagrammu, tā tiek izstrādāta transformējot TFM – funkcionālās īpašības kļūst par objektiem, bet topoloģiskās attiecības par saitēm starp tiem.
7.	Topoloģiskā klašu diagramma	3	Pakotņu, Stāvokļu un Objektu diagrammas	Topoloģiskā klašu diagramma attēlo problēmvides modeli un sistēmas struktūras modeli. Problēmvides modeļa galvenais mērķis ir definēt vizuālu abstrakciju vārdnīcu. Topoloģiskās attiecības starp klasēm parāda cēloņseku attiecības starp artefaktiem problēmvidē.
8.	Objektu diagramma	4	-	Objektu diagramma tiek izstrādāta Topoloģiskās klašu diagrammas bagātināšanas laikā, analizējot asociācijas starp klasēm. Tā ir noderīga gadījumos, kad viena un tā paša tipa objekti vienlaicīgi izpilda dažādas lomas. Objektu diagramma tiek lietota arī gadījumos, kad nepieciešams parādīt sistēmas piemēru kādā noteiktā laika punktā.
9.	Stāvokļu diagramma	5	-	Stāvokļu diagrammas tiek lietotas objektu stāvokļu pāreju analīzei un specificēšanai. Katram objekta tipam tiek definēta sava Stāvokļu diagramma.
10.	Pakotņu diagramma	6	Komponentu diagramma	Pakotņu diagrammā tiek apkopotas un sagrupētas klases atbilstoši apakšsistēmu apgabaliem un to nodrošinātajai funkcionalitātei.
11.	Komponentu diagramma	7	Izvietojuma diagramma	Komponentu diagrammā tiek apkopotas pakotnes no Pakotņu diagrammas, lai specificētu modulāras, izvietojamas un aizvietojamas sistēmas daļas. Vienai pakotnei tiek sagatavots viens komponents.
12.	Izvietojuma diagramma	8	-	Izvietojuma diagrammā tiek specificēts komponentu izvietojums skaitļošanas mezglos.

4.2. TopUML modelēšanas metodes aktivitātes

TopUML modelēšanas metodes process ir dots 4.2. attēlā Aktivitāšu diagrammas veidā, kur katra darbība parāda vienu modelēšanas aktivitāti, bet saites starp tām – darbību secību „no augšas uz leju” specificēšanas procesam.



4.2. att. TopUML modelēšanas metodes aktivitātes

Problēmvides analīze un lietojumvides projektēšana TopUML modelēšanas metodē tiek veikta, izpildot sešas aktivitātes (katra aktivitāte definē ieejas (nepieciešamos) un izejas (sagatavotos) artefaktus, kā arī to izstrādei veicamās darbības):

1. *Problēmvides funkcionēšanas analīze* – atbilstoši problēmvides funkcionēšanas īpašībām tiek sagatavots to atspoguļojošs TFM [16], kā arī nepieciešamās lietojumvides funkcionēšanu specificējošs TFM [15] un identificētas saistības starp izstrādātajām funkcionālajām īpašībām un programmatūras prasībām [18].
2. *Uzvedības analīze un projektēšana* – par pamatu izmantojot sagatavoto TFM tiek izstrādātas Topoloģiskā lietošanas gadījumu, Secību, Aktivitāšu un Mijiedarbības pārskata diagrammas. Topoloģiskā lietošanas gadījumu diagramma atspoguļo informāciju par apakšsistēmām, atbilstoši TFM noslēgšanas operāciju izpildei. [15][19]
3. *Struktūras analīze un projektēšana* – transformējot lietojumvides funkcionēšanu atspoguļojošo TFM sākotnēji tiek izstrādāta Komunikāciju diagramma un pēc tam Topoloģiskā klašu diagramma, kas ietver klases un topoloģiskās attiecības starp tām. Jāatzīmē, ka TFM transformācija nodrošina precīzu nepieciešamās funkcionalitātes noteikšanu katrai klasei. Veicot Topoloģiskās klašu diagramma bagātināšanu tā tiek papildināta ar citu veidu attiecībām starp klasēm (piemēram, asociācijām, vispārināšanu) un tiek izstrādātas Objektu diagrammas. [17][67]
4. *Stāvokļu izmaiņu un pāreju analīze* – stāvokļi un to pārejas tiek definētas ar Stāvokļu diagrammām. Stāvokļu diagrammu nepieciešams sagatavot un analizēt katram objektam, kas ietilpst sistēmas galvenajā funkcionēšanas ciklā (šie objekti ir sistēmai svarīgākie objekti). [20]
5. *Loģiskās struktūras projektēšana* – loģiskā struktūra tiek definēta ar Pakotņu diagrammas palīdzību, kur katra pakotne sākotnēji atspoguļo vienu apakšsistēmu. Pakotņu saturs tiek noteikts atbilstoši apakšsistēmā ietilpstošajiem lietošanas gadījumiem un to savstarpējām saistībām ar klasēm no Topoloģiskās klašu diagrammas. [19]

6. *Komponenšu un izvietojuma projektēšana* – šīs aktivitātes laikā tiek sagatavotas Komponenšu un Izvietojuma diagrammas, atbilstoši izstrādātajai Pakotņu diagrammai un definētajām nefunkcionālajām prasībām. [19]

4.3. Kopsavilkums

Problēmvides analīzei un lietojumvides projektēšanai TopUML modelēšanas metodē ir paredzētas sešas aktivitātes, kas pārklāj dinamikas, struktūras, izkārtojuma un izvietojuma modelēšanu. Veicot aktivitātes vienu pēc otras, kā definēts modelēšanas metodē, programmatūra tiek projektēta „no augšas uz leju" – sākot ar problēmvides formalizēšanu un beidzot ar projektēto komponentu izvietošanas plānošanu, kā arī visi izstrādātie artefakti ir definēti atbilstoši problēmvides funkcionēšanas īpašībām. Šādā veidā tiek nodrošināta cēloņseku identificēšana un definēšana, kā problēmvides, tā lietojumvides artefaktos. Ieguvums no šādas formalizētas modelēšanas metodes lietošanas ir programmatūras sagatavošana atbilstoši problēmvides funkcionēšanas īpašībām, problēmvides funkcionēšanas izmaiņu ietekmes novērtēšana lietojumvidē (un otrādi).

TopUML modelēšanas metode darba ietvaros ir salīdzināta ar citām UML valodas vadītām modelēšanas metodēm, izmantojot dažādus kritērijus, kas sagrupēti četrās grupās: metodē lietotie analīzes un projektēšanas modeļi, problēmvides analīzes un domēna modeļa izstrādes pieeja, prasību pārvaldība un metodes lietošana praksē. Tā kā TopUML modelēšana un TFMfMDA pieeja ir balstītas TFM un tā īpašību lietošanā, vairāku kritēriju novērtējums abām metodēm ir līdzīgs vai vienāds. Salīdzinājumā ar TFMfMDA pieeju, TopUML modelēšana atrisina vienu no tās vājākajām vietām – veicamo uzdevumu jeb atbildību noteikšanu identificētajām klasēm. TopUML modelēšanas metodē šis aspekts ir atrisināts, veicot TFM transformāciju par Topoloģisko klašu diagrammu, kā arī citu veidu diagrammām.

Kopsavilkumā jāatzīmē, ka TopUML profils un modelēšanas metode kopā atrisina identificētās nepilnības UML valodā un tās lietošanā programmatūras projektēšanā.

5. TOPUML IMPLEMENTĀCIJA UN APROBĀCIJA

TopUML profila un modelēšanas metodes implementācija un aprobācija ir parādīta, izmantojot divus programmatūras projektēšanas projektus:

1. *Biznesa atbalsta sistēmas projektēšana* – eksperimentāla programmatūras projektējuma izstrāde veļas mazgātavas problēmvidei,
2. *Organizācijas datu sinhronizācijas sistēmas izstrāde* – reālas programmatūras izstrāde datu iegūšanai no vairākiem datu avotiem un ievietošanai vienā datu glabātuvē, veicot detalizētu piemēra analīzi.

Par detalizēta piemēra analīzi tiek uzskatīta programmatūras izstrādes procesa novērošana un iegūto datu savākšana visa projekta laikā, savukārt eksperimentāla projektējuma izstrāde ir formāla un kontrolēta projektēšanas izpēte [26]. Katrs no apskatītajiem projektiem izmanto dažādas TopUML modelēšanas daļas un aspektus.

Piemēram, eksperimentā tiek lietoti sistēmas mērķi Secību diagrammas apjoma noteikšanai, bet reālās programmatūras izstrādē – funkcionālajām prasībām atbilstoši lietošanas gadījumi.

Šī nodaļa papildus ietver empīrisku TopUML profila un modelēšanas metodes novērtējumu, kuru ir veikušas divas ekspertu grupas. Abas ekspertu grupas piedalījās biznesa atbalsta sistēmas eksperimentālajā projektēšanā veļas mazgātavas problēmvidei [17]. Pirms dalības eksperimentā visi dalībnieki aizpilda zināšanu pašnovērtējuma anketu par modelēšanas metodēm, valodām un rīkiem, tādejādi ļaujot novērtēt priekšzināšanu ietekmi jaunas valodas un metodes apgūšanai.

5.1. Biznesa atbalsta sistēmas projektēšana

Biznesa atbalsta sistēmas projektēšana tiek demonstrēta praktiskā eksperimentā, analizējot veļas mazgātavas problēmvidi un projektējot tai atbilstošu lietojumvidi. Problēmvides funkcionēšanas apraksts ir sastādīts tādā veidā, lai uzskatāmi demonstrētu TopUML modelēšanas aktivitātes un profilā definēto diagrammu konstruēšanu. Eksperimentālās sistēmas projektēšanas laikā ir sagatavoti vairāki artefakti, atbilstoši TopUML modelēšanas metodei:

1. *Problēmvides* un *lietojumvides TFM* – izstrādāts saskaņā ar neformālo sistēmas darbības aprakstu un izvirzītajām funkcionālajām prasībām,
2. *Secību diagrammas* un *Mijiedarbības pārskata diagramma* – sagatavotas transformējot TFM, secību diagrammu apjoms noteikts atbilstoši sistēmas mērķiem,
3. *Komunikāciju diagramma* – veicot TFM transformāciju ir iegūti aktieri, objekti, saites starp tiem, nosūtāmie ziņojumi, kā arī ziņojumu secības numuri,
4. *Topoloģiskā klašu diagramma* – definēta transformējot TFM un Komunikāciju diagrammu, iegūstot diagrammu ar klasēm, to veicamajiem uzdevumiem un topoloģiskajām attiecībām starp tām.

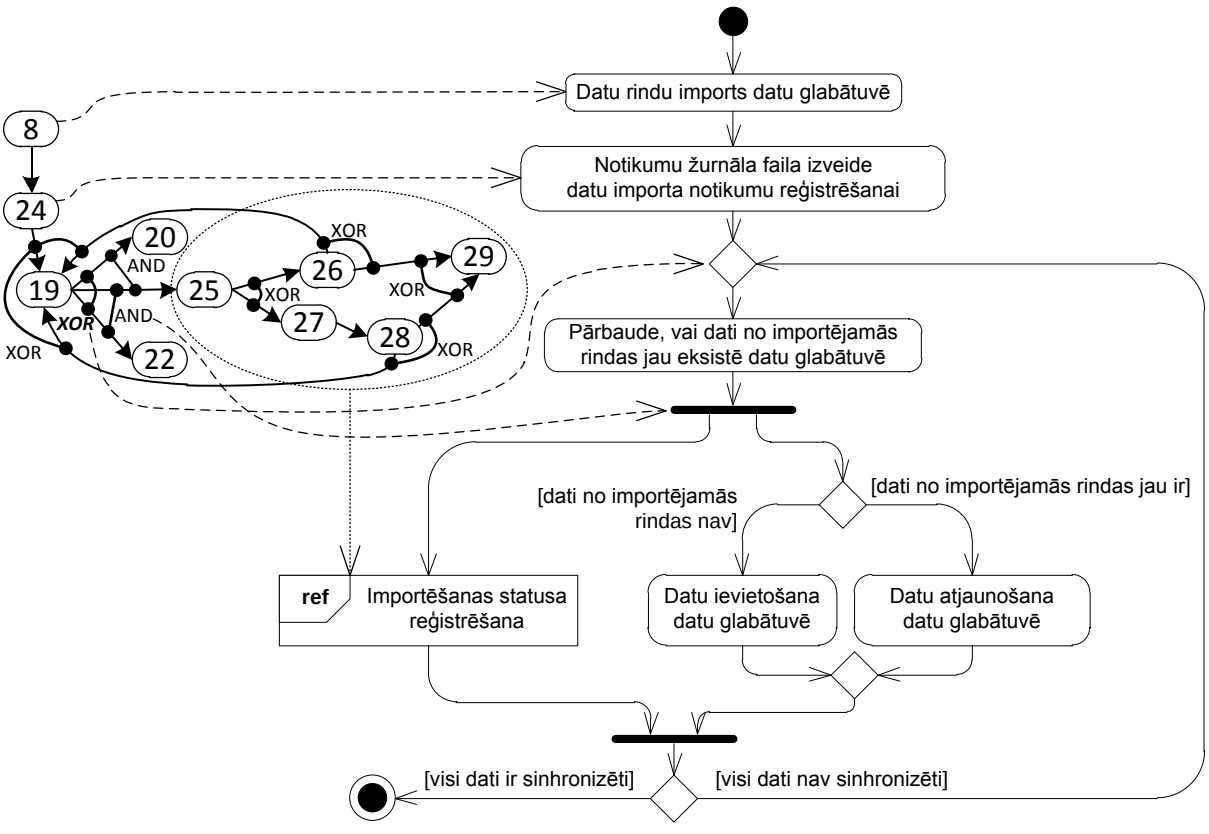
Visi projektēšanas laikā sagatavotie artefakti ir izstrādāti atbilstoši problēmvides funkcionēšanas īpašībām. Pateicoties TFM lietošanai problēmvides formalizēšanai un TFM īpašību pārnešanai uz citām TopUML diagrammām, izstrādātos artefaktus ir iespējams viennozīmīgi atsekot kā lietojumvidē, tā problēmvidē. Eksperimentālā piemēra izstrādes laikā izmantotā teorija un iegūtie rezultāti ir publicēti [21] un [69], kā arī metodiskajā līdzeklī [17].

5.2. Organizācijas datu sinhronizācijas sistēmas izstrāde

Organizācijas datu sinhronizācijas sistēmas projektēšanas un izstrādes projekta laikā TopUML profils un modelēšanas metode ir lietota reālas programmatūras izveidei SIA „Lattelecom Technology”. Paralēli ar projektēšanas un programmēšanas darbiem ir veikta detalizēta piemēra analīze, dokumentējot un novērtējot izstrādātos analīzes un projektējuma artefaktus. Izstrādātās programmatūras mērķis ir datu iegūšana no vairākiem datu avotiem un ievietošana vienā datu glabātuvē. Piemēra analīze aptver visu programmatūras dzīves ciklu, izstrādātā programmatūra funkcionē produkcijas vidē un ir pārcelta uz uzturēšanas fāzi.

Projekta realizācijas laikā ir sagatavoti vairāki artefakti, atbilstoši TopUML modelēšanas metodei:

1. *Problēmvides* un *lietojumvides TFM* – izstrādāts saskaņā ar neformālo sistēmas darbības aprakstu un izvirzītajām funkcionālajām prasībām,
2. *Topoloģiskā lietošanas gadījumu diagramma* – definēta atbilstoši izstrādātajam TFM un noteiktajām saistībām starp tajā ietilpstošajām funkcionālajām īpašībām un izvirzītajām funkcionālajām prasībām,
3. *Secību* un *Aktivitāšu diagrammas* – sagatavotas transformējot TFM, diagrammu apjoms noteikts atbilstoši lietošanas gadījumiem (piemērs TFM transformācijai par Aktivitāšu diagrammu ir dots 5.1. attēlā),
4. *Komunikāciju diagramma* – iegūta veicot TFM transformāciju,
5. *Sākotnējā* un *bagātinātā Topoloģiskā klašu diagramma* – sākotnējā diagramma definēta transformējot TFM un Komunikāciju diagrammu, savukārt bagātinātā diagramma iegūta veicot papildus problēmvides analīzi pēc TopUML modelēšanā definētajiem soļiem (rezultātā sākotnējā diagramma ir papildināta ar asociāciju, vispārināšanas un atkarību attiecībām starp klasēm, kā arī ar nodrošinātajiem un nepieciešamajiem interfeisiem atbilstoši TFM ieejām un izejām),
6. *Stāvokļu diagramma* – projekta ietvaros sagatavota viena Stāvokļu diagramma sistēmas galvenajam objektam (identificēts atbilstoši piederībai galvenajam funkcionēšanas ciklam), veicot TFM transformāciju.



5.1. att. TFM fragmenta transformācija par Aktivitāšu diagrammu, atbilstoši lietošanas gadījumam „Datu importēšana datu glabātuvē"

Pateicoties TopUML modelēšanas metodes lietošanai programmatūras sagatavošanas laikā izstrādātos artefaktus ir iespējams viennozīmīgi atsekot gan lietojumvidē, gan problēmvidē, tādejādi ļaujot precīzi novērtēt vienas vides izmaiņu ietekmi otrā vidē. Galvenie programmatūras izstrādes piemēra analīzes rezultāti ir publicēti [19], kā arī jaunas teorētiskās atziņas ir atspoguļotas [15] un [20].

5.3. Empīrisks TopUML profila un modelēšanas novērtējums

TopUML profila un modelēšanas metodes empīrisks novērtējums ir balstīts uz darba rezultātiem ar divām programmatūras izstrādes ekspertu darba grupām, veicot biznesa atbalsta sistēmas eksperimentālo projektēšanu veļas mazgātavas problēmvidei. Darbam ar ekspertiem ir izdalīti šādi aspekti:

1. *Eksperimentālās projektēšanas mērķis un process* – mērķis ir programmatūras projektēšana, izmantojot TopUML modelēšanas metodi, lai tādejādi pārbaudītu tās apmācīšanās iespējas un lietojamību. Projektēšanas process ir sadalīts atsevišķos astoņos darba semināros, katram seminārам definējot sagaidāmo rezultātu.
2. *Dalībnieki* – abās darba grupās kopā ir 32 dalībnieki, 15 no tiem bakalaura inženriezinātņu grāds datorzinātnē un 17 – maģistra, kā arī visiem dalībniekiem ir iepriekšēja pieredze programmatūras izstrādē, veicot dažādus pienākumus (analītiķis, programmētājs, testētājs un projekta vadītājs).
3. *Eksperimenta rezultāts* – ir sagatavoti vairāki sistēmas projektējumi, kuru sasvstarpējais salīdzinājums parāda formālas metodes un modeļu lietošanas priekšrocības (atšķirības starp konstruētajiem modeļiem ir minimālas), kā arī ir veikts piedāvātās modelēšanas metodes empīrisks novērtējums.
4. *Pozitīvais un negatīvais TopUML modelēšanā* – kā viens no eksperimenta rezultātiem ir tā dalībnieku vērtējums par pozitīvo (spēcīgs teorētiskais pamats, formālas aktivitātes, transformācijas starp modeļiem, samazināta iespējamība izstrādātā programmatūras koda pārstrādāšanai) un negatīvo (rīku trūkums) TopUML modelēšanas metodes lietošanā.

Apkopojot darba rezultātus ir sagatavots un publicēts metodiskais līdzeklis „Topoloģiskā biznesa sistēmu modelēšana un programmatūras sistēmu projektēšana” [17], kas ietver gan TopUML modelēšanas metodes teoriju, gan praktisku piemēru tās lietošanai problēmvides analīzē un lietojumvides projektēšanā.

5.4. Kopsavilkums

Veicot izstrādātā TopUML profila un modelēšanas metodes aprobāciju ir izstrādāti divi dažādi projekti: eksperimentāla sistēmas projektēšana veļas mazgātavas problēmvidei un reālas programmatūras izstrāde organizācijas datu sinhronizēšanas nodrošināšanai. Aprobēšanas laikā ir parādīts, kā, veicot definētos modelēšanas soļus un lietojot specificēto TopUML profilu, analīzes un projektējuma artefakti tiek sagatavoti atbilstoši problēmvides funkcionēšanas īpašībām. Vienlaikus tiek saglabātas cēloņseku sakarības kā problēmvides aprakstošajos elementos, tā lietojumvides elementos, kā arī starp tiem. Rezultātā tiek iegūta programmatūra, kas atbilst problēmvides funkcionēšanai un tās nosacījumiem, un cēloņseku sakarības, kas atvieglo turpmāku izstrādātās programmatūras uzturēšanu un attīstīšanu.

Pateicoties identificētajām trasējamības saistībām ir iespējams veikt detalizētu problēmvides funkcionēšanas izmaiņu ietekmes novērtēšanu lietojumvidē (un otrādi).

Modeļvadāmās arhitektūras kontekstā piedāvātā metode nodrošina no skaitļošanas neatkarīgā un platformneatkarīgā modeļa izstrādi, kā arī transformāciju veikšanu starp šiem modeļiem. Veicot programmatūras projektēšanu ar TopUML modelēšanas metodi tiek iegūta atbilde uz jautājumu „*Kas?*" – tā nosaka kas ir kas un kas katram no elementiem ir jāveic (piemēram, klašu un to veicamo uzdevumu noteikšana). Savukārt jautājums „*Kā?*" attiecas uz platformspecifisko līmeni, kas piedāvātajā metodē nav ietverts. Jautājums „*Kā?*" nozīmē kādā veidā veicamais uzdevums tiks realizēts konkrētā platformā vai tehnoloģijā (piemēram, kādā veidā dati tiek saglabāti datu bāzē).

DARBA REZULTĀTI UN SECINĀJUMI

Darba mērķis bija papildināt UML valodu ar teorētisko bāzi, lai radītu pamatu notācijas pārvēršanai par formālu modelēšanas valodu, un sagatavot modelēšanas metodi, kas ļauj viennozīmīgi izsekot cēloņseku sakarībām kā problēmvidē, tā lietojumvidē. Darba galvenais rezultāts ir TFM papildināšana ar loģiskajām attiecībām, TopUML profila specifikācijas un modelēšanas metodes definēšana, lai izstrādātā profila lietošanu ieviestu praksē. Visi izvirzītā darba mērķa sasniegšanai definētie ir izpildīti, iegūstot šādus rezultātus un secinājumus:

1) UML valodas, tās specifikācijas un modelēšanas metožu analīzes rezultāti ir šādi:
   a) neskatoties uz ieguvumiem lietojot UML valodu programmatūras izstrādē, tās specifikācijas un lietošanas analīze uzrāda vairākas šīs valodas nepilnības,
   b) lai gan UML valoda sākot ar versiju 2.0 ietver tās paplašināšanas mehānismu, profilu izstrādi apgrūtina UML kā notācijas specifikācija, specificējot tikai valodas elementus un to semantiku,
   c) analizējot vairākus profilus ir gūta pārliecība, ka piemērotākais veids profila specifikācijas izstrādei ir pēc iespējas tās struktūru veidot atbilstoši UML valodas specifikācijas struktūrai,
   d) dažādu modelēšanas metožu analīzes rezultāts parāda, ka modelēšanas valodas lietošanu praksē nosaka tieši modelēšanas metodes un pieejas, nevis pati valoda; līdz ar to daļu no identificētajām UML valodas nepilnībām ir iespējams risināt ar atbilstošas modelēšanas metodes palīdzību,
   e) diemžēl daļējais programmatūras dzīvescikla pārklājums un fragmentārais UML diagrammu lietojums analizētajās modelēšanas metodēs UML valodas nepilnības nenovērš pietiekamā apjomā.
2) Augstāk minētie analīzes rezultāti noved pie secinājuma, ka izvirzītā darba mērķa sasniegšanai nepieciešams:
   a) papildināt UML valodu ar teorētisko bāzi, tādejādi sagatavojot TopUML profila specifikāciju,
   b) definēt efektīvu un lietojamu modelēšanas metodi TopUML profila izmantošanai programmatūras izstrādē.

3) TFM papildināšana ar loģiskajām attiecībām dod nepieciešamos līdzekļus šī modeļa transformēšanai uz cita veida diagrammām ar sarežģītu struktūru.
4) Papildinot UML valodu ar TFM matemātiskās topoloģijas formālismu, ir iegūta tāda modelēšanas valodas specifikācija, kas izveido pamatu notācijas pārvēršanai par formālu modelēšanas valodu un kas satur pietiekamus elementus viennozīmīgai cēloņseku attiecību identificēšanai, specificēšanai un izsekošanai.
5) Savukārt, iekļaujot piedāvātajā modelēšanas metodē tādus aspektus, kā pienācīgu problēmvides un lietojumvides analīzi, pēc iespējas lielāku profila diagrammu un programmatūras izstrādes dzīvescikla pārklājumu, tiek samazināti un pat novērsti tādi identificētie UML valodas un tās lietošanas trūkumi, kā izmērs, sarežģītība, nesaskanība un neviennozīmīgums.
6) Lietojot TFM kā saknes modeli citu diagrammu sintēzē, formālā veidā tiek panākts šāds rezultāts:
   a) visi lietojumvidei sagatavotie artefakti atbilst problēmvides funkcionēšanas īpašībām,
   b) viennozīmīga cēloņseku attiecību izsekojamība starp izstrādātajiem artefaktiem gan vienā, gan dažādos abstrakcijas līmeņos,
   c) sagatavotie artefakti veic tikai tiem paredzētos uzdevumus,
   d) izstrādātie komponenti nav ciešā sasaistē ar pārējo sistēmu un tiem ir definēta nepieciešamā saskarne jeb interfeisi.
7) Izstrādātais TopUML profils un tam paredzētā modelēšanas metode ir veiksmīgi aprobēta gan eksperimentālā programmatūras projektēšanā, gan reālā programmatūras izstrādes projektā.
8) Veiktā aprobācija parāda, ka TopUML modelēšanā artefaktu definēšana modeļvadāmās arhitektūras kontekstā tiek veikta no skaitļošanas neatkarīgā un platformneatkarīgā modeļa līmenī.
9) TopUML profila un modelēšanas metodes aprobācija problēmvides analizēšanā un sistēmas projektēšanā kopā ar divām ekspertu grupām sniedz empīrisku novērtējumu piedāvātajai modelēšanas valodai un metodei. Kā ieguvums piedāvātajai metodei ir atzīmēts tās teorētiskais pamats, formālās modelēšanas aktivitātes, transformācijas starp modeļiem, kas kopā samazina iespējamību izstrādātā programmatūras koda pārstrādāšanai.

Turpmākie iespējamo pētījumu virzieni ir šādi:

- TopUML profilu atbalstoša modelēšanas rīka sagatavošana,
- ar TopUML profilu sagatavoto modeļu transformēšana no platformneatkarīgā uz platformspecifisko modeli, kā arī programmatūras koda ģenerēšana.

BIBLIOGRĀFISKAIS SARAKSTS

[1] Alksnis G. Formal Specification Languages and Category Theory Within the Framework of MDA// Computer Science, Applied Computer Systems, Vol.26, Nr.5, Scientific Proceedings of Riga Technical University, Riga, Latvia: RTU Publishing, 2006. - pp. 33-41

[2] Ambler S. Elements of UML 2.0 Style. - New York, USA: Cambridge University Press, 2005. - 200 p.

[3] Arlow J., Neustadt I. UML 2 and the Unified Process: Practical Object-Oriented Analysis and Design. - Upper Saddle River, NJ, USA: Addison-Wesley, 2nd ed., 2005. - 624 p.

[4] Asnina E. Formalization of Problem Domain Modeling within Model Driven Architecture. Doctoral thesis. - Riga, Latvia: RTU Publishing house, 2006. - 195 p.

[5] Asnina E. The Formal Approach to Problem Domain Modelling Within Model Driven Architecture// Proceedings of the 9th International Conference "Information Systems Implementation and Modelling" (ISIM'06), Přerov, Czech Republic: Jan Štefan MARQ, 2006. - pp. 97-104

[6] Asnina E., Gulbis B., Osis J., Alksnis G., Donins U., Slihte A. Backward Requirements Traceability within the Topology-based Model Driven Software Development// Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development. - Beijing, China: SciTePress, 2011. - pp. 36-45

[7] Batra D., Satzinger J. Contemporary Approaches and Techniques for the Systems Analyst// Journal of Information Systems Education. - 2006. - 17(3) - pp. 257–265

[8] Booch G. Object Oriented Analysis and Design with Applications. - Upper Saddle River, NJ, USA: Addison-Wesley, 2nd ed., 1993. - 608 p.

[9] Booch G., Maksimchuk R., Engel M., Young B., Conallen J., Houston K. Object-oriented analysis and design with applications. - Upper Saddle River, NJ, USA: Addison-Wesley, 3rd ed., 2007. - 720 p.

[10] Booch G., Rumbaugh J., Jacobson I. The Unified Modeling Language User Guide. - Upper Saddle River, NJ, USA: Addison-Wesley, 2nd ed., 2005. - 475 p.

[11] Breu R., Hinkel U., Hofmann C., Klein C., Paech B., Rumpe B., Thurner V. Towards a Formalization of the Unified Modeling Language // ECOOP'97 – Object-Oriented Programming, 11th European Conference (Lecture Notes in Computer Science, Vol. 1241). – Berlin, Germany: Springer, 1997. - pp. 344-366

[12] Burton-Jones A., Meso P. Conceptualizing Systems for Understanding: An Empirical Test of Decomposition Principles in Object-Oriented Analysis// Information Systems Research. - 2006. - 17(1) - pp. 38-60

[13] DeLoach S., Hartrum T. A Theory-Based Representation for Object-Oriented Domain Models// IEEE Transactions on Software Engineering. - 2000. - Volume 26, Issue 6. - pp. 500-517

[14] Dobing B., Parsons J. Dimensions of UML Diagram Use: Practitioner Survey and Research Agenda// Principle Advancements in Database Management Technologies: New Applications and Frameworks. – Hershey, New York, USA: Information Science Reference, 2010. - pp. 271-290

[15] Donins U. Semantics of Logical Relations in Topological Functioning Model// Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012) – 2012. (To be published)

[16] Donins U. Software Development with the Emphasis on Topology// Advances in Databases and Information Systems (Lecture Notes in Computer Science, Vol.5968). - Berlin, Germany: Springer-Verlag, 2010. - pp. 220-228

[17] Doniņš U. Topological business systems modeling and software systems design. - Riga, Latvia: RTU Publishing house, 2011. - 65 p. (in Latvian)

[18] Donins U., Osis J. Reconciling Software Requirements and Architectures within MDA// Scientific Proceedings of Riga Technical University, Computer Science (Series 5), Applied Computer Systems (Vol. 38). - Riga, Latvia: RTU Publishing house, 2009. - pp. 84-95

[19] Donins U., Osis J. Topological Modeling for Enterprise Data Synchronization System: A Case Study of Topological Model-Driven Software Development// Proceedings of the 13th International Conference on Enterprise Information Systems, Volume 3. - Beijing, China: SciTePress, 2011. - pp. 87-96

[20] Donins U., Osis J., Asnina E., Jansone A. Formal Analysis of Objects State Changes and Transitions// Proceedings of the 7th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2012) – 2012. (To be published)

[21] Donins U., Osis J., Slihte A., Asnina E., Gulbis B. Towards the Refinement of Topological Class Diagram as a Platform Independent Model// Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development. - Beijing, China: SciTePress, 2011. - pp. 79-88

[22] Erickson J., Siau K. Theoretical and Practical Complexity of Modeling Methods// Communications of the ACM. - 2007. - 50(8) - pp. 46-51

[23] Evans A., Kent S. Core Meta-Modelling Semantics of UML: The pUML Approach// «UML»'99: The Unified Modeling Language. Beyond the Standard (Lecture Notes in Computer Science, Vol. 1723). - Berlin, Germany: Springer, 1999. - pp. 140-155

[24] Evermann J., Wand Y. Ontological Modeling Rules For UML: An Empirical Assessment// Journal of Computer Information Systems. - 2006. - 46(5) - pp. 14-29

[25] Evermann J., Wand Y. Towards Ontologically-Based Semantics for UML Constructs// Conceptual Modeling – ER 2001, 20th International Conference on Conceptual Modeling (Lecture Notes in Computer Science, Vol. 2224). - Berlin, Germany: Springer, 2001. - pp. 341-354

[26] Fenton N., Pfleeger S. Software Metrics: A Rigorous and Practical Approach. - Scottsdale, Arizona, USA: Coriolis Group, 2nd ed., 1996. – 649 p.

[27] Fowler M. Why use the UML?// Software Development. - 1998. - Volume 6, Issue 3

[28] Fowler M. UML Distilled: A Brief Guide to the Standard Object Modeling Language. - Upper Saddle River, NJ, USA: Addison-Wesley, 3rd ed., 2003. - 208 p.

[29] Grundspenkis J. Fault Localisation Based on Topological Feature Analysis of Complex System Model// Diagnostics and Identification. - Riga: Zinatne, 1974. - pp. 38-48 (in Russian)

[30] Grundspenkis J. Structural Modelling of Complex Technical Systems in Conditions of Incomplete Information: A Review// Modern Aspects of Management Science. - Riga, Latvia: RTU Publishing house, 1997. - pp. 111-136

[31] Grundspenkis J. Structural Modelling with ASMOS in the Early Stages of Design// Software for Manufacturing. – Amsterdam, Holland: North-Holland Publishing Company, 1989. - pp. 229-239

[32] Grundspenkis J. The Synthesis and Analysis of Structure in Computer Aided Design// Computer Applications in Production and Engineering: Proceedings of the First International Conference. – Amsterdam, Holland: North-Holland Publishing Company, 1983. - pp. 301-316

[33] Grundspenkis J., Blumbergs A. Investigation of Complex System Topological Model Structure for Analysis of Failures// Issues of Technical Diagnosis. - Rostov-on-Don, Russia: Rostov Institute of Building Engineering, 1981. - pp. 41-48 (in Russian)

[34] He X. Formalizing UML Semantics// 25th Annual International Computer Software and Applications Conference (COMPSAC'01). - Chicago, Illinois, USA: IEEE Computer Society, 2001. - pp. 277

[35] International Organization for Standardization (ISO): ISO/IEC/IEEE 42010:2011 "Systems and software engineering -- Architecture description", 2011. - 37 p.

[36] Jacobson I., Christerson M., Jonsson P., Overgaard G. Object-Oriented Software Engineering: A Use Case Driven Approach. - Upper Saddle River, NJ, USA: Addison-Wesley, 1992. - 552 p.

[37] Jones C. Positive and Negative Innovations in Software Engineering// International Journal of Software Science and Computational Intelligence. - 2009. - Volume 1, Issue 2. - pp. 20-30

[38] Karpics I., Markovics Z. Development and Evaluation of Normal Performance Recovery Method of a Functional System // Proceedings of 9th IEEE International Symposium on Applied Machine Intelligence and Informatics (SAMI), 2011, Slovakia, Smolenice, 2011. - pp. 171-175

[39] Kent S. The Unified Modeling Language// Formal Methods for Distributed Processing: A Survey of Object-Oriented Approaches. - Cambridge, England: Cambridge University Press, 2001. - pp. 126-151

[40] Kim S., Carrington D. Formalizing the UML Class Diagram Using Object-Z// «UML»'99: The Unified Modeling Language. Beyond the Standard (Lecture Notes in Computer Science, Vol. 1723). - Berlin, Germany: Springer, 1999. - pp. 83-98

[41] Kleppe A., Warmer J., Bust W. MDA Explained. The Model Driven Architecture: Practice and Promise. - Upper Saddle River, NJ, USA: Addison-Wesley, 2003. - 192 p.

[42] Kobryn C. UML 2001: A Standardization Odyssey// Communications of the ACM. - 1999. - Volume 42, Issue 10. - pp. 29-37

[43] Lano K., Kolahdouz-Rahimi S. Model-Driven Development of Model Transformations// Theory and Practice of Model Transformations (Lecture Notes in Computer Science, Volume 6707). - Berlin, Germany: Springer-Verlag, 2011. - pp. 47-61

[44] Larman C. Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development. - Upper Saddle River, NJ, USA: Prentice Hall, 3rd ed., 2005. - 736 p.

[45] Lazar I., Motogna S., Parv B., Lazar C. Realizing Use Cases for Full Code Generation in the Context of fUML// Proceedings of the 2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development - Portugal: SciTePress, 2010. - pp. 80-89

[46] Li D., Li X., Stolz V. QVT-Based Model Transformation using XSLT// ACM SIGSOFT Software Engineering Notes. - 2011. - Volume 36, Issue 1. - pp. 1-8

[47] Loton T. UML Software Design with Visual Studio 2010. – Breinigsville, PA, USA: LOTONtech Limited, 2010. - 136 p.

[48] Markovica I., Markovics Z. Mathematical Model of Pathogenesis of Hard Differentiable Diseases// Cybernetics and Diagnostics, Volume 4. - Riga: Zinatne, 1970. - pp. 21-28 (in Russian)

[49] Mellor S., Balcer M. Executable UML: A Foundation for Model-Driven Architecture. - Upper Saddle River, NJ, USA: Addison-Wesley, 2002. - 416 p.

[50] Mens T., Van Gorp P. A Taxonomy of Model Transformation// Electronic Notes in Theoretical Computer Science. - 2006. - Volume 152. - pp. 125-142

[51] Miller J., Mukerji J. (editors): MDA Guide Version 1.0.1 / Internet. - http://www.omg.org/cgi-bin/doc?omg/03-06-01.pdf

[52] Nielsen M., Havelund K., Wagner K., George C. The RAISE Language, Method and Tools// Formal Aspects of Computing. - 1989. - Volume 1 Issue 1. - pp 85-114

[53] Ņikiforova O. System Modeling in UML with Two-Hemisphere Model Driven Approach// Scientific Proceedings of Riga Technical University, Computer Science (Series 5), Applied Computer Systems (Volume 43). - Riga, Latvia: RTU Publishing house, 2010. - pp. 37-44

[54] OMG: Meta Object Facility (MOF) Core Specification Version 2.0 / Internet. - http://www.omg.org/spec/MOF/2.0/PDF/

[55] OMG: Service Oriented Architecture Modeling Language (SoaML) / Internet. - http://www.omg.org/spec/SoaML/1.0/Beta2/PDF

[56] OMG: Unified Modeling Language Infrastructure Version 2.4.1 / Internet. - http://www.omg.org/spec/UML/2.4.1/Infrastructure/PDF/

[57] OMG: Unified Modeling Language Superstructure Version 2.4.1 / Internet. - http://www.omg.org/spec/UML/2.4.1/Superstructure/PDF/

[58] OMG: OMG Systems Modeling Language (OMG SysML) / Internet. - http://www.omg.org/spec/SysML/1.2/PDF

[59] Olive A. Conceptual Modeling of Information Systems. - Berlin, Germany: Springer, 2007. - 455 p.

[60] Osis J. Extension of Software Development Process for Mechatronic and Embedded Systems// Proceeding of the 32nd International Conference on Computer and Industrial Engineering. - Limerick, Ireland: University of Limerick, 2003. - pp. 305-310

[61] Osis J. Formal Computation Independent Model within the MDA Life Cycle// International Transactions on Systems Science and Applications. - 2006. - Volume 1, Number 2. - pp. 159-166

[62] Osis J. Mathematical Description of Complex System Functioning// Cybernetic and Diagnosis, Volume 4. - Riga: Zinatne, 1970. - pp. 7-14 (in Russian)

[63] Osis J. The Topological Model of System Functioning// Automatics and Computer Science, Volume 6. - Riga, Latvia, 1969. - pp. 44-50 (in Russian)

[64] Osis J., Asnina E. Enterprise Modeling for Information System Development within MDA// Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008). - Chicago, Illinois, USA: IEEE Computer Society, 2008. - pp. 491

[65] Osis J., Asnina E. Model-Driven Domain Analysis and Software Development: Architectures and Functions. – Hershey, New York, USA: IGI Global, 2011. - 487 p.

[66] Osis J., Donins U. An Innovative Model Driven Formalization of the Class Diagrams// Proceedings of the 4th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2009). – Portugal: INSTICC Press, 2009. - pp. 134-145

[67] Osis J., Donins U. Formalization of the UML Class Diagrams// Evaluation of Novel Approaches to Software Engineering (Communications in Computer and Information Science (CCIS), Volume 69). - Berlin, Germany: Springer-Verlag, 2010. - pp. 180-192

[68] Osis J, Donins U. Modeling Formalization of MDA Software Development at the Very Beginning of Life Cycle// Advances in Databases and Information Systems. 13th East-European Conference, ADBIS 2009: Associated Workshops and Doctoral Consortium, Local Proceedings. - Riga, Latvia: JUMI Publishing House Ltd., 2009. - pp. 48-61

[69] Osis J., Donins U. Platform Independent model Development by Means of Topological Class Diagrams// Proceedings of the 2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development - Portugal: SciTePress, 2010. - pp. 13-22

[70] Osis J., Gefandbein J., Markovitch Z., Novozhilova N. Diagnosis based on graph models. (By the Examples of Aircraft and Automobile Mechanisms). - Moscow, Russia: Transport, 1991. - 244 p. (in Russian)

[71] Osis J., Silins J. Topological Function-Architecture Co-Design of Embedded Systems// Advances in Databases and Information Systems. 13th East-European Conference, ADBIS 2009: Associated Workshops and Doctoral Consortium, Local Proceedings. - Riga, Latvia: JUMI Publishing House Ltd., 2009. - pp. 424-431

[72] Osis J., Šlihte A. Transforming Textual Use Cases to a Computation Independent Model// Proceedings of the 2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development - Portugal: SciTePress, 2010. - pp. 33-42

[73] Osis J., Sukovskis U., Teilans A. Business Process Modeling and Simulation Based on Topological Approach// Proceedings of the 9th European Simulation Symposium and Exhibition. - Passau, Germany, 1997. - pp. 496-501

[74] Owre S., Rushby J., Shankar N. PVS: A Prototype Verification System// 11th International Conference on Automated Deduction (Lecture Notes in Artificial Intelligence, Volume 607). -Berlin, Germany: Springer, 1992. - pp. 748-752

[75] Pardillo J. A Systematic Review on the Definition of UML Profiles// Model Driven Engineering Languages and Systems (Lecture Notes in Computer Science, Volume 6394). - Berlin, Germany: Springer-Verlag, 2010, - pp. 407-422

[76] Podeswa H. UML for the IT Business Analyst. - Boston, MA, USA: Course Technology PTR, 2nd ed., 2009. - 372 p.

[77] Rumbaugh J., Blaha M., Premerlani W., Eddy F. Lorensen W. Object-Oriented Modeling and Design. - Englewood Cliffs, NJ: Prentice Hall, 1991. - 528 p.

[78] Rumbaugh J., Jacobson I., Booch G. The Unified Modeling Language Reference Manual. - Upper Saddle River, NJ, USA: Addison-Wesley, 2nd ed., 2004. - 721 p.

[79] Scott K. The Unified Process Explained. - Upper Saddle River, NJ, USA: Addison-Wesley, 2001. - 208 p.

[80] Sejans, J., Nikiforova, N. Practical Experiments with Code Generation from the UML Class Diagram// Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development. - Beijing, China: SciTePress, 2011. - pp. 57-67

[81] Siau K., Cao Q. Unified Modeling Language (UML) - a Complexity Analysis// Journal of Database Management. - 2001. - Volume 12, Issue 1. - pp. 26-34

[82] Siau K., Cao, Q. How Complex Is the Unified Modeling Language?// Advanced Topics in Database Research. - 2002. - Volume 1. - pp. 294-306

[83] Siau K., Loo P. Identifying Difficulties in Learning UML// Information Systems Management. - 2006. -Volume 23, Issue 3. - pp. 43-51

[84] Simons A., Graham I. 37 Things that Don't Work in Object-Oriented Modeling with UML// Proceedings of ECOOP 98 Workshop on Precise Behavioral Semantics. - Universitat Muchen, 1998. - pp. 209-232

[85] Slihte A., Osis J., Donins U. Knowledge Integration for Domain Modeling// Proceedings of the 3rd International Workshop on Model-Driven Architecture and Modeling-Driven Software Development. - Beijing, China: SciTePress, 2011. - pp. 46-56

[86] Slihte A., Osis J., Donins U., Asnina, E., Gulbis, B. Advancements of the Topological Functioning Model for Model Driven Architecture Approach// Proceedings of the 3rd

International Workshop on Model-Driven Architecture and Modeling-Driven Software Development. - Beijing, China: SciTePress, 2011. - pp. 91-100

[87] Spivey J. The Z Notation: A Reference Manual. - Prentice Hall, 2nd ed., 1992. - 150 p.

[88] Stevens P., Pooley R. Using UML: Software Engineering with Objects and Components. - Harlow, England: Addison-Wesley, 2nd ed., 2005. - 250 p.

[89] Szlenk M. UML Static Models in Formal Approach// Balancing Agility and Formalism in Software Engineering (Lecture Notes in Computer Science, Volume 5082). - Berlin, Germany: Springer-Verlag, 2008. - pp. 129-142

[90] Turner M. Microsoft Solutions Framework Essentials: Building Successful Technology Solutions. - Redmond, Washington, USA: Microsoft Press, 2006. - 300 p.

[91] Warmer J., Kleppe A. The Object Constraint Language: Getting Your Models Ready for MDA. - Upper Saddle River, NJ, USA: Addison-Wesley, 2nd ed., 2003. - 240 p.

[92] Xueming L., Parsons J. Ontological Semantics for the Use of UML in Conceptual Modeling// ER (Tutorials, Posters, Panels & Industrial Contributions). - 2007. - pp. 179-184

[93] Zhao Y., Zong-Yuan Y., Xie J. Pi-Calculus Based Assembly Mechanism of UML State Diagram and Validation of Model Refinement// Proceedings of International Conference on Electronic Computer Technology 2009 (ICECT 2009). - Macau, China, 2009. - pp 604-609