RĪGAS TEHNISKĀ UNIVERSITĀTE

Gundars ALKSNIS

KATEGORIJU TEORIJAS PIELIETOŠANA FORMĀLO SPECIFIKĀCIJU VALODU INTEGRĒŠANAI MODEĻU VADĀMĀ ARHITEKTŪRĀ

Promocijas darba kopsavilkums

Rīga 2008

RĪGAS TEHNISKĀ UNIVERSITĀTE
Datorzinātnes un informācijas tehnoloģijas fakultāte
Lietišķo datorsistēmu institūts

Gundars ALKSNIS
Datorsistēmu doktora programmas doktorants

KATEGORIJU TEORIJAS PIELIETOŠANA FORMĀLO SPECIFIKĀCIJU VALODU INTEGRĒŠANAI MODEĻU VADĀMĀ ARHITEKTŪRĀ

Promocijas darba kopsavilkums

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

RTU IZDEVNIECĪBA
Rīga 2008

UDK 004.2+512.58](043.2)
Al 514 k

Alksnis G. Kategoriju teorijas pielietošana formālo specifikāciju valodu integrēšanai modeļu vadāmā arhitektūrā. Promocijas darba kopsavilkums. – R.:RTU, 2008. – 31 lpp.

Iespiests saskaņā ar DITF LDI institūta 2007. gada 16. maija lēmumu, protokols Nr.54.

Šis darbs izstrādāts ar Eiropas Sociālā fonda atbalstu Nacionālās programmas "Atbalsts doktorantūras programmu īstenošanai un pēcdoktorantūras pētījumiem" projekta "Atbalsts RTU doktorantūras attīstībai" ietvaros.

ISBN 978-9984-32-818-8

PROMOCIJAS DARBS
IZVIRZĪTS RĪGAS TEHNISKAJĀ UNIVERSITĀTĒ
INŽENIERZINĀTŅU DOKTORA GRĀDA IEGŪŠANAI (datorsistēmās)

Promocijas darbs inženierzinātņu doktora grāda (datorsistēmās) iegūšanai tiek publiski aizstāvēts 2008. gada 9. jūnijā Rīgas Tehniskās universitātes Datorzinātnes un informācijas tehnoloģijas fakultātē, Meža ielā 1/3, 202. auditorijā.

OFICIĀLIE OPONENTI:

Dr.sc.math., profesors Kārlis Šadurskis
Rīgas Tehniskā universitāte, Latvija

Dr.habil.sc.comp., profesors Jānis Bārzdiņš
Latvijas universitāte, Latvija

Dr.habil.sc.ing., profesors Zbigņevs Huzārs
Vroclavas Tehnoloģijas universitāte, Polija

APSTIPRINĀJUMS

Es 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.

Gundars Alksnis ..... (Paraksts)

Datums: .....

Promocijas darbs ir uzrakstīts latviešu valodā, satur ievadu, 6 nodaļas, nobeigumu, literatūras sarakstu, 36 attēlus un 1 tabulu, kopā 144 lapaspuses. Literatūras sarakstā ir 81 literatūras avots.

1. VISPĀRĪGS DARBA RAKSTUROJUMS

1.1. Promocijas darba tēmas aktualitāte

Modeļu izstrāde un modelēšana programmatūras izstrādes dzīves cikla sākuma posmos mūsdienās tiek ļoti plaši pielietota, jo šādā veidā labāk var izprast projektējamo sistēmu, un laicīgi pieņemt kvalitatīvus lēmumus par tās tālāko realizāciju.

Parasti modeļi tiek attēloti diagrammu veidā, kas balstās uz noteiktu grafisko notāciju. Vislielāko izplatību starp šādām notācijām, neapšaubāmi, ir ieguvusi *vienotā modelēšanas valoda* UML [62]. Vēl vairāk, pēdējos gados ir aktualizējusies ideja par atbilstošu rīku realizāciju, kas ļautu veidot sintaktiski un semantiski pilnīgus sistēmu modeļus, un programmatūras koda ģenerēšana no modeļiem būtu pilnībā automatizēta. Respektīvi, modeļi un to transformācijas, izvirzās par galveno artefaktu programmatūras izstrādē. Vienu no šādām uz modeļiem balstītām programmatūras izstrādes vīzijām piedāvā OMG konsorcijs. To sauc par *modeļu vadāmo arhitektūru* MDA [61].

MDA pamatā ir viedoklis, ka programmatūras izstrādē galvenais process ir sistēmas modeļu projektēšana, uzlabošana un automātiska transformēšana līdz pat izpildāmam kodam. Tas pamatojas uz UML standartu pielietošanu. Kaut arī šī vīzija izskatās daudzsološa un programmatūras izstrādes akadēmiskajās aprindās un industrijā jau ir iegūta veiksmīga pieredze tās pielietošanā, tomēr, pēc darba autora domām, no formalizācijas viedokļa šī pieeja joprojām ir nepilnīga. Proti, modeļu transformācijās, kas tiek uzskatītas par būtiskāko aspektu, nav izstrādāts pilnībā formalizēts teorētiskais pamatojums.

MDA ideju publicēšana datējama ar 2001. gadu, tomēr tās praktiskā realizācija joprojām nav pilnībā nobriedusi — modeļu realizācijas un transformāciju iespējas mūsdienās piedāvātajos rīkos lielākoties ir daļējas, un modeļus nepieciešams papildināt, veicot papildus programmēšanu. Darba autors ir pārliecināts, ka šīs vīzijas sasniegšanas būtiskākais šķērslis ir grafisko tehnoloģiju nepilnīgā semantika, kas bieži vien noved pie pretrunīgām modeļu interpretācijām, izraisot projektējumu kļūdas jau programmatūras izstrādes sākuma posmos.

Protams, tendence ir šos trūkumus mazināt vai novērst, tomēr arī tekošajā UML 2.0 versijā diagrammu formālā semantika joprojām nav strikti specificēta (vai arī definēta neformāli). Pēc darba autora domām, UML semantikas specifikācija arī nākotnē, visticamāk, paliks tāda, ka UML modeļus varēs izmantot, gan lai atspoguļotu neformālus (t.s. skices), gan arī formālus (t.i. viennozīmīgi aprakstošus) UML modeļus. Tomēr, vēlams, lai vismaz UML modeļu formālā semantika būtu UML kopējās semantikas iespēju apakškopa.

Skatoties no citas puses, programminženierijā ir izstrādātas metodes, kurās programmatūras izstrāde pilnībā balstās uz matemātiski precīzi formulētiem principiem — tās sauc par *formālām metodēm*. Katrai formālai metodei ir piesaistīta formāla matemātiskā notācija, ar kuras palīdzību tiek rakstītas sistēmu *formālās specifikācijas*. Tāpat, formālo metožu pielietošanas pētījumi ir ar senāku priekšvēsturi nekā UML un MDA, un ir izstrādātas dažādas *formālo specifikāciju valodas*, kas tiek pielietotas datorsistēmu specificēšanai. Tomēr praktiski tas notiek tikai specifiskās nozarēs, kas saistītas ar kritiskām vai iegultām sistēmām, kurās nekorekta sistēmas uzvedība vai kļūdas var novest pie būtiskiem zaudējumiem, t.sk. letāliem (piemēram, medicīnā, aviācijā, kodoltehnoloģijās, u.c.).

Piedāvātajā promocijas darbā tiek izvirzīta sekojoša hipotēze. Ja neformālos grafiskos

modeļus varētu transformēt vai papildināt ar atbilstošu formālu notāciju, kas ir piemērota formālai analīzei (piemēram, formālās specifikācijās, kurām ir ne tikai semantika, bet arī var definēt funkcionālas attiecības starp specifikācijas elementiem), tad ar šādiem modeļiem varētu veikt manipulācijas (piemēram, izmantot teorēmu pierādītājos un veikt modeļu pārbaudes), lai formāli pārliecinātos par modeļu un to transformāciju korektumu un viennozīmīgumu.

Hipotēzes pārbaudīšanai, par grafisko (UML) un formālo notāciju sasaistes teorētisko pamatu tiek izmantoti *kategoriju teorijas* [63] pamatprincipi. Kategoriju teorija ir matemātiska valoda ar plašu pielietošanas spektru [34]. Ar tās palīdzību var formāli abstrahēties no pētāmo struktūru iekšējām sastāvdaļām, visu uzmanību veltot tikai to savstarpējām attiecībām, piemēram, lai formāli aprakstītu objektu kompozīciju un uzlabošanu (*refinement*), neatkarīgi no pašu objektu iekšējām īpašībām.

Līdz ar to **darba aktualitāte** ir saistīta ar esošajiem UML semantikas trūkumu novēršanas jautājumiem — galvenokārt, lai formalizētu ar UML palīdzību izstrādātos modeļus, kā arī lai veicinātu formālo metožu izmantošanu programmatūras izstrādē, tajās industrijas sfērās, kur nepieciešama kvalitatīva programmatūra, bet kuras nav tik kritiskas, lai formālu metožu pielietošana un pierādījumu izvešana būtu obligāts sistēmas izstrādes priekšnosacījums.

1.2. Darba mērķis

Darba mērķis ir izpētīt esošās UML semantikas formalizācijas pieeju trūkumus un dot risinājumus formālo metožu pielietošanai UML neformālo īpašību un dinamiskās semantikas trūkumu novēršanai, kā arī pielietot formālās metodes modeļu transformācijām posmā starp modeļu vadāmas arhitektūras *platformneatkarīgiem modeļiem* PIM un *platformspecifiskiem modeļiem* PSM.

Kā risinājums UML neformālo īpašību un dinamiskās semantikas trūkumu novēršanai un modeļu formālo pārbaužu veikšanai, tiek izmantotas formālās metodes, kā arī kategoriju teorijas vispārināšanas koncepcijas formālo specifikāciju valodu integrēšanai modeļu vadāmā arhitektūrā prasību specificēšanas un projektēšanas posmos PIM modeļos.

1.3. Darba uzdevumi

Lai sasniegtu darba mērķi, tika izvirzīti sekojoši uzdevumi:

- Izmantot kategoriju teorijas vispārināšanas koncepcijas formālo specifikāciju valodu integrēšanai UML modeļos modeļu vadāmās arhitektūras kontekstā, prasību specificēšanas un projektēšanas posmos.
- Izpētīt daļējo specifikāciju integrācijas iespējas kopējā datorsistēmas specifikācijā, izmantojot kategoriju teorijas principus modeļu formalizēšanai ar formālo notāciju palīdzību.
- Aprobēt piedāvātos risinājumus UML stāvokļu mašīnu modeļu gadījumā, un pielietot praktiska uzdevuma atrisināšanai.
- Izstrādāt ieteikumus modeļu pārbaužu veikšanai ar UML stāvokļu mašīnu modeļiem.

Pētījuma objekts ir UML modeļu un formālo metožu notāciju semantika.

Pētījuma subjekts ir kategoriju teorijas izmantošana UML semantikas un formālo metožu sasaistīšanai programmatūras izstrādes MDA vīzijā.

1.4. Pētījumu metodika

Darba teorētiskie pētījumi pamatojas uz matemātisku formālismu pielietošanu datorsistēmu prasību formālai definēšanai un analīzei. Teorētiskie rezultāti pamatojas uz kopu teoriju, datu tipu teoriju un kategoriju teoriju. Darbā demonstrētās sistēmas *TraSer* projektēšanā un izstrādē izmantotas vispārpieņemtas programminženierijas metodes.

1.5. Zinātniskā novitāte

Darba zinātniskais jaunieguvums pamatojas uz trīs koncepciju (formālo metožu notāciju, vienotās modelēšanas valodas UML un kategoriju teorijas) pielietošanu, ar mērķi integrēt formālo metožu notācijas modeļu vadāmās arhitektūras UML modeļos. Darbā piedāvātais risinājums veicina formālismu ieviešanu un izmantošanu tajās industrijas sfērās, kuras nav kritiskas, bet, to pielietojot, var uzlabot datorsistēmu kvalitāti un pārvaldību.

Promocijas darbā piedāvātais formālo specifikāciju valodu UML modeļu integrācijas ietvars risina galveno ar UML saistīto modeļu vadāmo pieeju trūkumu — formāla matemātiskā pamatojuma ieviešanu. Promocijas darbs apliecina, ka var veiksmīgi apvienot gan ar UML saistītās modeļu vadāmās arhitektūras, gan formālās metodes, demonstrējot formālo pieeju lietderību, ne tikai kritiskajām datorsistēmām, bet arī plašākās IKT sfērās, kurās formālo metožu pielietošana pašlaik nav pietiekoši izplatīta.

1.6. Galvenie rezultāti

Darba galvenais rezultāts ir formālo specifikāciju valodu un UML modeļu integrācijas ietvars, kas ļauj integrēt formālās notācijas modeļu vadāmas arhitektūras programmatūras izstrādes procesā, par formālo pamatojumu izmantojot kategoriju teorijas morfismu un funktoru definīciju pielietojumus.

Cits darba rezultāts ir UML stāvokļu mašīnu modeļu pārbaužu metožu analīze un pielietošanas iespēju novērtēšana.

1.7. Praktiskais pielietojums

Darba rezultātu praktiskais pielietojums ir saistīts ar formālo metožu izmantošanas veicināšanu programmatūras izstrādes industrijā arī nekritisku datorsistēmu projektēšanā, ar mērķi paaugstināt mūsdienās arvien pieaugošas sarežģītības sistēmu kvalitāti. Piedāvātais integrācijas ietvars izmantojams arī kritisku un iegultu sistēmu projektēšanas procesā, jo UML modeļi nodrošina viegli uztveramus sistēmas modeļus, kamēr formālās specifikācijas nodrošina to viennozīmīgumu.

1.8. Publikācijas

Darba **galvenie pētījumu rezultāti** ir publicēti sekojošās starptautiskās konferencēs un semināros:

1. Alksnis G. The Analysis of UML State Machine Formal Checking Methods// Proceedings of the 10th International Conference Information System Implementation and Modeling (ISIM'07). Eds: A. Kelemenová, D. Kolář, etc. - Opava: Silesian University, 2007. - pp. 131–136.

2. Alksnis G. UML stāvokļu mašīnu modeļu formālās pārbaudes metožu salīdzinoša analīze// Scientific Proceedings of Riga Technical University. Computer Science, Applied Computer Systems, Vol.26, Nr.5 - Riga: RTU Publishing, 2007. - pp. 28–37.

3. Alksnis G. Formal Methods and Model Transformation Framework for MDA// Proceedings of the 1st International Workshop on Formal Models (WFM'06). Eds. D. Kolář and A. Meduna. - Ostrava: MARQ, 2006. - pp. 87–94.

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

5. Alksnis G., Asnina E., Osis J., Silins J. Formalization of Software Development: Problems and Solutions// Scientific Proceedings of Riga Technical University. Computer Science, Applied Computer Systems, Vol.22, Nr.5 - Riga: RTU Publishing, 2005. - pp. 204–216.

6. Alksnis G. Formal Specification from Category Theory Viewpoint// Scientific Proceedings of Riga Technical University. Computer Science, Applied Computer Systems, Vol.17, Nr.5 - Riga: RTU Publishing, 2003. - pp. 137–144.

7. Alksnis G., Osis J. Formalization of Software Engineering by Means of the Theory of Categories// Scientific Proceedings of Riga Technical University, Computer Science, Applied Computer Systems, Vol.13, Nr.5 - Riga: RTU Publishing, 2002. - pp. 157–163.

8. Alksnis G. Category Theory in Software Engineering// Proceedings of the 5th International Baltic Conference (DB&IS 2002), Vol.1 - Tallinn: Institute of Cybernetics at TTU, 2002. - pp. 269–278.

9. Alksnis G., Osis J. Kategoriju teorija datorzinātnēs// Scientific Proceedings of Riga Technical University. Computer Science, Applied Computer Systems, Vol.8, Nr.5 - Riga: RTU Publishing, 2001. pp. 59–67.

1.9. Darba struktūra un apjoms

Promocijas darbs sastāv no ievada, sešām nodaļām, nobeiguma un literatūras saraksta.

Darba *ievada* daļā tiek izklāstīta pētījuma motivācija, formulēti pētījumu mērķi un to sasniegšanai izpildāmie uzdevumi.

Darba *pirmā nodaļa* veltīta formālo metožu un to notāciju apskatam un novērtēšanai, ar mērķi tās pielietot datorsistēmu formālu specifikāciju izstrādē un analīzē. Tiek aplūkota formālo specifikāciju valodu klasifikācija, bet detalizētāk tiek aplūkotas sekojošas formālās notācijas — Z, Maude un TLA$^+$. Katra no tām pārstāv savu formālo valodu klasi un ir piemērota noteiktu datorsistēmas aspektu specificēšanā un verificēšanā.

Darba *otrā nodaļa* dod ieskatu kategoriju teorijā, tās attīstībā, un promocijas darbā izmantoto kategorijas teorijas koncepciju izklāstam un demonstrācijai. Ar piemēriem tiek demonstrētas šīs teorijas pielietošanas sfēras un specifika datorzinātnes un programminženierijas kontekstā.

Trešajā nodaļā tiek detalizēti analizētas darba pirmajā nodaļā aplūkotās formālo specifikāciju valodas no kategoriju teorijas viedokļa. Tiek demonstrēts, ka aplūkotajām

formālo specifikāciju valodām, noteiktā abstrakcijas līmenī, ir kopējas īpašības, kas ļauj veikt šo īpašību analīzi, un, kas ir būtiski, šīs notācijas savstarpēji integrēt.

Darba *ceturtā nodaļa* ir veltīta UML modeļu pārbaudēm ar mērķi veikt formālas pārbaudes pirms datorsistēmas modeļi tiek transformēti programmēšanas valodu kodā — tas ļauj iegūt papildus informāciju par izstrādāto modeli, kā arī atklāt un novērst modeļa nepilnības. Tiek apskatītas četras UML stāvokļu mašīnu modeļu pārbaužu metodes, un veikta šo metožu savstarpējā analīze un salīdzinājums.

Darba *piektā nodaļa* veltīta promocijas darba galvenā rezultāta izklāstam. Šajā nodaļā tiek izklāstīts piedāvātais UML modeļu un formālo notāciju integrācijas ietvars, kas ir paredzēts izmantošanai modeļu vadāmās arhitektūrā. Tiek izklāstīta piedāvātā ietvara būtība un definēti formālo notāciju metamodeļi, kas pēc tam tiek integrēti UML klašu un stāvokļu mašīnu diagrammās.

Sestajā nodaļā ar praktiski realizētas datorsistēmas *TraSer* projektējuma piemēru, tiek demonstrēti piektajā nodaļā izklāstītā ietvara pielietojuma principi. Šīs sistēmas pielietošanas sfēra ir izkliedētu transakciju atbalsta nodrošināšana, kas, izvirza noteiktas kvalitātes prasības, kuru nodrošināšanai tiek pielietots darbā piedāvātais ietvars.

Promocijas darba *nobeiguma* daļā tiek dots iegūto rezultātu apkopojums, doti secinājumi un norādīti turpmāko pētījumu attīstīšanas virzieni.

2. PROMOCIJAS DARBA IZKLĀSTS

Turpmākajās apakšnodaļās tiek dots konspektīvs promocijas darba izklāsts ar uzsvaru uz darba rezultātiem un secinājumiem.

2.1. Formālās metodes

Promocijas darba pirmā nodaļa ir veltīta formālo metožu un to notāciju apskatam un novērtēšanai, ar mērķi tās pielietot datorsistēmu formālu specifikāciju izstrādē un analīzē.

2.1.1. Formālo metožu loma programminženierijā

Būtiskākās ar datorsistēmu izstrādi saistītās problēmas ir tās, ka to izstrāde bieži vien prasa vairāk laika, izmaksā vairāk nekā plānots un rezultāts neapmierina pasūtītāju. Tā sauktā *programmatūras krīze* pirmo reizi tika identificēta 1960-to gadu beigās [60], un kopš tā laika šīs problēmas novēršanai ir izstrādāti un piedāvāti daudz dažādu risinājumu, tomēr problēma, pēc būtības, joprojām ir aktuāla.

Tomēr, lai kāds būtu risinājums, viens aspekts paliek nemainīgs — jo vēlāk programmatūras izstrādes procesā tiek atklātas kļūdas vai neatbilstības prasībām, jo vairāk resursu un lielākas izmaksas nepieciešamas, kļūdu novēršanai.

Daļa no piedāvātajiem risinājumiem aizstāv viedokli, ka inženiertehnisku problēmu analīzē un risināšanā, nepieciešams pielietot atbilstošus matemātiskus principus un (formālas) metodes. Datorzinātnē šāda tehnoloģijas dēvē par *formālām metodēm*. Katrai formālai metodei ir noteikta un viennozīmīga matemātiskā notācija, ar kuras palīdzību tiek definēti likumi *formālām specifikācijām*.

Svarīgi ir tas, ka formālajās metodēs matemātika tiek pielietota, nevis ir pētījuma objekts, t.i. esošās matemātikas aksiomas un pierādītās teorēmas tiek pieņemtas par patiesām.

Diemžēl lielākā daļa mūsdienu programminženierijas projektēšanas un dokumentēšanas metožu, lai aprakstītu realizējamo datorsistēmu, izmanto neformālas vai pusformālas pieejas — tiek izmantotas dabiskā valodā rakstītas specifikācijas, kuras papildina ar dažādām grafiskām diagrammām, kas palīdz labāk saprast projektējamo datorsistēmu. Tomēr šo metožu būtiskākais trūkums ir tāds, ka tās nenodrošina pēc iespējas efektīvāku projektējumu un viennozīmīgi interpretējamu dokumentāciju.

Formālo notāciju priekšrocība ir tā, ka tās ir precīzas un viennozīmīgi interpretējamas. Pareizi pielietojot, tiek nodrošināta datorsistēmas specifikācija, kurā nav pretrunu un divdomību.

2.1.2. Formālo specifikāciju valodu apskats

Mūsdienās ir klasificētas vairākas formālo specifikāciju valodu klases. Katru klasi pārstāvošā formālā notācija paredzēta specifisku datorsistēmas aspektu aprakstīšanai. Ir izdalītas sekojošas formālo specifikāciju valodu klases [81]:

- Uz modeļiem balstītās — datorsistēma tiek aprakstīta kā modelis, izmantojot tādas matemātiskās konstrukcijas kā kopas un secības, bet ar operācijām nosakot datorsistēmas stāvokļu maiņas nosacījumus. Šīs grupas atpazīstamākais pārstāvis ir Z notācija [18], kas darbā tiek apskatīta detalizētāk. Arī B metodes abstraktas mašīnas notācija pieder šai valodu klasei [1], bet Z notācijai funkcionāli vislīdzīgākā ir VDM metode [44], kurā lai gan nav pieejamas shēmas, tomēr uzsvars tiek likts uz uzlabošanas procesu;
- Algebriskās — datorsistēma tiek aprakstīta ar abstraktu datu tipu kopā ar šo tipu operācijām un attiecībām. Visplašāk pazīstamā valoda ir OBJ [39], un tās daudzie paveidi. Viens no tās paveidiem ir *Maude*, kas darbā tiek aplūkota detalizētāk. Cita šīs klases pārstāve ir Specware Slang [74], kurai ir cieša integrācija ar kategoriju teorijas principiem;
- Loģiskās — orientējas uz pirmās vai augstākas kārtas loģiku. Visbiežāk tiek izmantotas teorēmu pierādīšanai. Piemēram, TLA$^+$, kas realizē temporālo loģiku [48]. Šīs valodas iespējas darbā tiek aplūkotas detalizētāk.
- Procesu orientētās — tiek izmantotas, lai specificētu datorsistēmas procesus un komunikāciju starp tiem (piemēram, PSL [66], CCS [58], CSP [42]);

2.1.3. Datorsistēmas modeļa specificēšana Z notācijā

Formālo specifikāciju valoda Z ir standartizēta (ISO/IEC 13568:2002) notācija, kas balstās uz pirmās kārtas predikātu loģiku, *Zermelo-Fränkel* aksiomātisko kopu teoriju, λ-rēķiniem un plašu pieraksta notāciju [75, 18].

Datorsistēmu Z notācijā modelē ar abstraktu sākuma stāvokli, un kā tam sekojošu operāciju secībām, kas modificē sākotnējo un tam sekojošos stāvokļus. Z notācija ir veidota tā, lai būtu izteiksmīga un saprotama, nevis izpildāma.

Tāpat kā lielākai daļai formālo notāciju, Z notācijā var definēt un izmantot: tipus, izteikumu loģiku, predikātus un kvantorus, aksiomātisko kopu teoriju, attiecības un funkcijas, secības. Tomēr raksturīgā Z notācijas pazīme ir shēmu notācija, kas ir galvenais mehānisms lielu specifikāciju strukturēšanai. Shēma sastāv no divām daļām — *signatūras* (mainīgie ar tiem piesaistītiem tipiem) un *predikāta*.

Z notācijai ir izstrādāti vairāki paplašinājumi, to skaitā objektorientētu specifikāciju rakstīšanai, piemēram, Object-Z [71, 25].

2.1.4. Abstrakto datu tipu specificēšana ar Maude

Algebrisko specifikāciju valodas OBJ principus 1976. gadā piedāvāja Dž. Gogēns [39]. Laika gaitā tā tika vairākkārt papildināta ar jaunām iespējām. Mūsdienās ar OBJ jau apzīmē valodu kopu, kas satur daudzas dažādas sākotnējās versijas modifikācijas un uzlabojumus. Pēdējās versijas pieņemtais nosaukums ir OBJ3, un viena tās dialektu realizācijām ir realizēta rīkā un notācijā Maude [23].

Maude balstās uz algebriskās programmēšanas teorijām — abstraktu datu tipu un to operāciju specificēšanu, pirmās kārtas vienādojumu loģiku, bet ir papildināta ar pārrakstīšanas (*rewrite*) un slēpto vienādojumu (*hidden equational*) loģikām, kā arī nodrošina efektīvu moduļu sistēmu parametrizētai programmēšanai.

Specifikācija valodā Maude sastāv no moduļu definīcijām un attieksmēm starp tiem. Jauni moduļi var iekļaut vai paplašināt jau esošos moduļus. Maude realizācija sastāv no divām daļām: bāzes (*Core*) un pilnās (*Full*) Maude. Pilnā Maude iekļauj bāzes Maude iespējas, bet tai ir papildus atbalsts objektorientētu moduļu specificēšanai, kas tiek detalizētāk pielietots promocijas darbā.

2.1.5. TLA$^+$ temporālās loģikas pielietošana formālajās specifikācijās

Formālo specifikāciju valodas TLA$^+$ sākotnējās idejas 1980-to gadu beigās piedāvāja L. Lamports [48]. Valodas nosaukums ir saīsinājums no *Temporal Logic of Actions*. Šīs valodas pamatā ir vienkāršota Amira Pnueli temporālā loģika [65], kuras mērķis ir aprakstīt sistēmu uzvedību. Patiesībā vairums TLA$^+$ specifikāciju sastāv no netemporālas matemātikas — temporālo loģiku var izmantot tikai tad, kad tā tik tiešām ir nepieciešama, lai atspoguļotu sistēmas uzvedību.

TLA$^+$ var izmantot plaša diapazona sistēmu specificēšanai — no programmu moduļu interfeisiem, līdz pat izkliedētām sistēmām, lai formāli aprakstītu praktiski visu diskrēto sistēmu veidus. Sevišķi piemērota tā ir asinhronu sistēmu aprakstīšanai — t.i. tādu sistēmu, kurām nav strikti noteikts soļa izpildes ilgums.

TLA$^+$ definē abstrakcijas, lai specificētu sekojošas klasiskās temporālās loģikas īpašības.

Drošuma īpašība (*safety property*) — apgalvojums tam, kas nedrīkst notikt. Definē programmas stāvokļu ierobežojumus.

Dzīvotspējas īpašība (*liveness property*) — apgalvojums tam, ka kaut kam obligāti jānotiek. Ļauj aizsargāt programmu no tā, lai programma neapmierinātu tikai sākotnējos nosacījumus, bet arī pārietu citas darbības izpildē. Dzīvotspējas īpašības izskata ar temporālās loģikas formulu palīdzību, un tām visu laiku ir jābūt spēkā.

Rakstot TLA$^+$ specifikācijas, vispirms izvēlas mainīgos, definē tipa invariantu un sākotnējā stāvokļa predikātu. Pēc tam specificē nākošā stāvokļa darbības, kas parasti aizņem specifikācijas lielāko daļu. Tikai pēc tam raksta specifikācijas temporālo daļu. Ja nepieciešams specificēt dzīvotspējas īpašības, jāizvēlas atbilstošs godīguma nosacījumus. Pēc tam vienā temporālās formulas definīcijā kombinē sākotnējo predikātu, nākošā stāvokļa darbību un izvēlētos godīguma nosacījumus. Šī formula arī veido TLA$^+$ specifikāciju.

2.1.6. Kopsavilkums

Šajā promocijas darba nodaļā apskatītās formālās notācijas ir tikai dažas no plašā formālo specifikāciju valodu klāsta. Apskatīto notāciju izvēle balstījās uz šo valodu

piemērotību integrēšanai modeļu vadāmās arhitektūras ietvarā.

Jebkura formāla notācija bez attiecīgā atbalstošā rīka praktiski nav lietderīga, jo formālo notāciju galvenais mērķis ir iegūt viennozīmīgi interpretējamas specifikācijas, līdz ar to notāciju sintakses un semantikas pārbaudēs, kā arī pierādījumu izvešanai, rīkiem ir būtiska loma. Būtībā noteiktas formālās notācijas praktiskas pielietošanas lietderība ir atkarīga no pieejamo rīku praktiskās lietderības. Tomēr jāņem vērā, ka ne visas formālās notācijas ir uzskatāmas par izpildāmām datora izpratnē — kaut arī tās ir viennozīmīgi interpretējamas, tās ir nedeterminētas.

Galvenie secinājumi ir sekojoši:

- Formālajās metodēs tiek pielietotas matemātiskas konstrukcijas;
- Matemātikas izmantošana ļauj pielietot papildus pārbaudes, kā arī novērš divdomības, kuras bieži rodas neformālām specifikācijām;
- Praktiska šādu metožu pielietošana programmatūras projektēšanā, ja korekti tiek pielietota, tad ļauj samazināt izdevumus, jo visas nepilnības un kļūdas tiek novērstas jau agrīnos projektēšanas posmos.

Neviena formālā notācija nav universāla — katra ir veidota noteiktu uzdevumu vai problēmu aprakstīšanai, līdz ar to tās izvēle balstās arī uz to, no kādiem aspektiem projektējamo sistēmu ir nepieciešams modelēt.

2.2. Kategoriju teorija datorzinātnēs

Promocijas darba otrā nodaļa dod ieskatu kategoriju teorijā, tās attīstībā, un promocijas darbā izmantoto kategorijas teorijas koncepciju izklāstam un demonstrācijai.

2.2.1. Kategoriju teorijas attīstība

Kategoriju teorija (KT) ir matemātikas nozares formāla valoda, kuras pirmsākumi meklējami 1942. gadā [29]. Proti, radās nepieciešamība formalizēt aprakstu, kā pāriet no vienām matemātikas struktūrām uz citām.

Ar 1980-tajiem gadiem sākās aktīvi pētījumi KT teorijas pielietojumiem datorzinātnē. Sevišķi aktīvi pētījumi bija algebriskās semantikas un funkcionālo programmēšanas valodu teorijas pētīšanā. Vēlāk KT tiek plaši pielietota dažādu formalizāciju analīzei formālajās projektēšanas metodēs.

KT konstrukciju pielietošana, ļauj formalizēt dažādas datorzinātnes problēmsfēras [2, 4, 33, 63, 80]. Šī formalizācija tiek panākta nevis uz dažādu ierobežojumu un standartu rēķina, bet gan uz problēmas vispārināšanas abstraktākās koncepcijās, kas, sākotnēji šķietami nesaistītām sfērām, ļauj atrast kopējas īpašības, vai vēl vairāk — pārnest problēmu no vienas problēmsfēras uz citu, kurā risinājums ir vieglāk atrodams, pēc tam šo risinājumu attiecinot uz avota problēmsfēru. Pateicoties KT formalizācijai un matemātiskajam pamatojumam, iespējams realizēt rīkus, kas balstās uz KT konstrukcijām un pielietot tos programminženierijā.

Diemžēl, KT pielietošanai ir arī pretinieki. Viņu argumenti ir sekojoši. Viens ir šķietamā vispārinātība, kurā kāda teorija vai piemērs tiek vispārināts tiktāl, ka patiesībā vairs nesatur nekādu praktisku interesi. Cits ir kategoriskā pārmērība, kādā KT tiek pielietota, problēmām, kas neprasa tik abstraktu atspoguļošanu vai terminoloģiju.

KT pamatdefinīcijas definē kategoriju, tās objektu un objektu morfismus.

Kategorija $\mathcal{C}$ ir orientēts grafs C kopā ar divām funkcijām $c : C_2 \to C_1$ un $u : C_0 \to C_1$, kur C_2 ir kopa, kas satur grafa ceļus ar garumu 2, C_1 — kopa, kas satur grafa lokus, kurus sauc par morfismiem, bet C_0 ir kopa, kas satur grafa virsotnes, kuras tiek sauktas par objektiem.

Lai saprastu pētāmo struktūru, nepieciešams saprast morfismus, kas to saglabā. No KT viedokļa, morfismi ir daudz svarīgāki par pašiem objektiem (un to iekšējo uzbūvi), jo tieši morfismi atspoguļo to, kas patiesībā ir pētāmā struktūra. Būtībā galvenais KT pētījumu objekts ir morfismi jeb attiecības starp objektiem.

Cits būtisks kategoriju teorijas termins ir *funktors*. Funktors F no kategorijas $\mathcal{C}$ uz kategoriju $\mathcal{D}$ ir grafu homomorfisms, kurš saglabā identitātes un kompozīcijas attiecības. Respektīvi, tas saglabā kategorijas struktūru. Funktoru viens no galvenajiem pielietojumiem ir aprakstīt transformācijas no vienas matemātikas disciplīnas uz citu.

Datorzinātnē funktoru pielietojums ir analoģisks — pārveidot konstrukcijas no vienas kategorijas uz citu, kurā problēmas risinājums ir vieglāk atrodams. Atrodot risinājumu, tas automātiski attiecinās arī uz sākotnējo kategoriju.

Visspilgtākā analoģija funktoriem ir ar MDA modeļu transformāciju definīcijām. Šajā gadījumā ir iesaistīti avota un mērķa modeļi, kuriem pēc strikti definētiem likumiem un noteikumiem, tiek veikta avota modeļa transformācija mērķa modelī, saglabājot avota modeļa strukturālās īpašības, kā tas ir transformācijās no PIM uz PSM modeļiem.

2.2.2. Kategoriju teorijas formālismu pielietojumi datorzinātnē

Kategoriju teorijas pielietošanai datorzinātnē ir izdalītas vismaz divas pieejas. Klasiskā pieeja, kuru dēvē par *tipu teorijas* vai *funkcionālo* pieeju, datorzinātni aplūko kā skaitļošanas zinātni ar uzsvaru uz rēķināšanu [15]. Proti, morfismi iekapsulē aprēķinus vai to abstrakcijas, kas no avota objekta ļauj iegūt mērķa objektu.

Pie šīs pieejas pieskaitāmas tādas tēmas kā λ-rēķinu [14], rekursiju teorijas [13] un (funkcionālo) programmēšanas valodu projektēšanas un semantikas [67] analīze no KT viedokļa.

Savukārt otra pieeja pēta datorsistēmu projektēšanu, un ir cieši saistīta ar programminženieriju [31]. Šajā pieejā morfismi tiek aplūkoti kā attiecības starp projektējamās sistēmas komponentēm, moduļiem, programmām un tamlīdzīgiem artefaktiem. Piemēram, uzlabošana no augsta līmeņa specifikācijām līdz izpildāmam programmas kodam. Tieši šī pieeja tiek izmantota promocijas darbā piedāvātā ietvara pamatojumam.

2.2.3. Kategoriju teorija un modeļu vadāmā arhitektūra

KT pielietošanas iespējas ir ļoti plašas, un pastāv daudzas problēmas, kas vēl nav līdz galam izanalizētas. Viena no tām ir mūsdienās tik aktuālā modeļu vadāmās arhitektūras koncepciju praktiska realizēšana.

No vienas puses, modeļu vadāmās arhitektūra pamatojas uz tādiem standartiem kā UML, *Meta Object Facility* (MOF), *Object Constraint Language* (OCL), *Query/Views/Transformations* (QVT), *Common Object Request Broker Architecture* (CORBA) un citiem, kas ļauj veidot un apmainīties ar modeļiem, kā arī no tiem automātiski ģenerēt programmas kodu. No otras puses, šie standarti nav pilnībā formāli, tāpēc pastāv interpretācijas iespējas, kas neveicina modeļu vadāmas programmatūras izstrādes ideju attīstību.

Viens no veidiem, kā novērst modeļu elementu divdomības, ir ieviest papildus formālas notācijas, kas sistēmas modeli viennozīmīgi paskaidrotu. Risinājums, ko iesaka darba autors ir integrēt formālo specifikāciju notācijas ar UML modeļu elementiem, tādā veidā viens otru papildinot — UML izmantojot, lai iegūtu viegli uztveramu priekšstatu par projektējamo sistēmu, savukārt, formālas notācijas — lai viennozīmīgi definētu sistēmas specifikācijas.

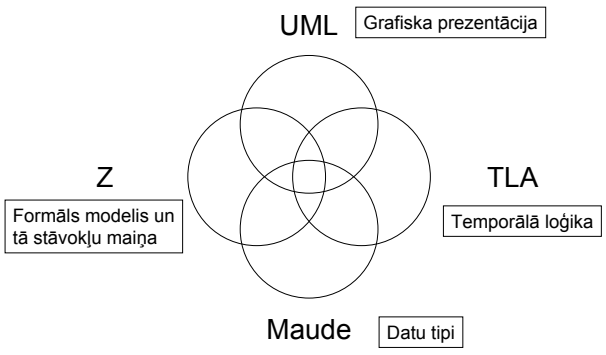
Šim mērķim KT var izmantot, lai šīs dažādās formālās notācijas savstarpēji integrētu, un pēc tam, savukārt, integrētu modeļu vadāmās programmatūras izstrādes procesā. Tieši šis aspekts vēl nav pietiekoši detalizēti izpētīts. KT izmantošana šim risinājumam balstās uz transformāciju pieejas pielietošanu — skaitļošanas pieeja šajā gadījumā nav piemērota, jo ar objektiem tiek uztverti sistēmas modeļi, bet morfismi ir to savstarpējās attiecības, kas apraksta, kā viens modelis ir cita modeļa sastāvdaļa, vai kā viens modelis attēlojas citā.

Pateicoties KT konstrukciju izmantošanai, var definēt universālu ietvaru, kas nav atkarīgs no konkrētām formālām notācijām, kamēr vien tās ir sasaistītas pašu notāciju modeļu (t.i. metamodeļu) līmenī. Respektīvi, notāciju semantika savstarpēji nepārklājas, bet ir apgabali, kas starp šīm notācijām ir kopēji, kā tas shematiski parādīts 1. attēlā. Izmantojot šos pārklājošos apgabalus, notācijas var savstarpēji integrēt. Tie apgabali, kas nepārklājas, specificē noteiktu sistēmas viedokli, piemēram, datu struktūru, procesu, sistēmas stāvokļu maiņu laikā, un citus. Bet visi kopā veido vienotu sistēmas specifikāciju.

2.2.4. Kopsavilkums

Apskatot būtiskākās KT pamatkoncepcijas, to pielietojums tiek demonstrēts ar ilustrējošiem piemēriem no datorzinātnes. Šī apskata mērķis ir dot priekšstatu par KT būtību un to pielietošanas iespējām dažādu formālu struktūru aprakstīšanai un īpašību analīzei. Galvenie KT principu pielietošanas aspekti ir sekojoši:

- KT ir matemātikas nozare, kas pēta matemātisku struktūru īpašības, apskatot nevis pētāmo objektu uzbūvi, bet gan to saistību (morfismiem) ar citiem šīs struktūras objektiem;
- Ar KT palīdzību var atrast kopsakarības starp dažādām nesaistītām struktūrām (kategorijām), un ar atbilstošu funktoru palīdzību šīs struktūras sasaistīt, tādā veidā ļaujot izvēlēties piemērotāko no šīm struktūrām, un pēc tam risinājumu attiecinot arī uz saistītajām struktūrām;



1. att. Formālo notāciju semantisko domēnu pārklājumu shēma

- Pateicoties šīm īpašībām, KT ir atrasti ļoti plaši pielietošanas apgabali, tai skaitā arī datorzinātnē. Ņemot vērā, ka datorzinātne, salīdzinājumā ar citam fundamentālajām zinātnēm ir jauna, tā ir vāji organizēta un fragmentāra — tai nav nostabilizējušies teorētiski pamatprincipi. Ar kategoriju teorijas palīdzību var vispārināt noteiktas datorzinātnes definīcijas un teorijas, un veikt to analīzi saistībā ar citām līdzīgām koncepcijām, tādā veidā tās savstarpēji sasaistot;

- Ar kategoriju teorijas konstrukciju palīdzību var veiksmīgi abstrahēties no datorzinātnei raksturīgās sīkumainības un detaļām, kas ļauj pētāmo koncepciju aplūkot plašākā aspektā.

2.3. Formālo specifikāciju integrācija

Promocijas darba trešajā nodaļā tiek apskatītas iespējas pielietot kategoriju teoriju formālo specifikāciju valodu analīzei un savstarpējai integrācijai, ar mērķi izdalīt un analizēt šo valodu sasaistošās īpašības.

2.3.1. Formālās notācijas no kategoriju teorijas viedokļa

KT ir piemērota, lai ar morfismu un funktoru palīdzību, attēlotu specifikāciju kompozīciju un uzlabošanu [5, 9, 10]. Līdz ar to, kategoriju teorijas konstrukcijas ir lietderīgi izmantot, lai analizētu formālo specifikāciju valodu kopīgās īpašības un savstarpējo saistību. Šādai analīzei sekojoša specifika:

- Sistēmas aprakstu uzdod kā savienotu loģisku teoriju kolekciju, katru definētu ar saviem tipiem, operācijām, aksiomām un teorēmām;

- Pēc tam sistēmas aprakstiem pielieto progresīvu uzlabošanu, sākot no prasību līmeņa, līdz pat izpildāmam kodam.

Galvenās universālās KT konstrukcijas, ar kurām tiek manipulētas specifikāciju struktūras: sākuma un gala objekti, izgrūdums un izvilkums, līdzrobeža un robeža, līdzpilnība un pilnība. Tā rezultātā visi aspekti tiek aprakstīti tieši un pilnībā formalizēti — gan pašas teorijas, gan attiecības starp tām (morfismi), gan teoriju grupu mijiedarbība (diagrammas), gan to uzlabošana.

Vēl vairāk, visām formālo notāciju klasēm var izdalīt kopējas īpašības, izsakot tās kategoriju teorijas terminos. Tā, piemēram, algebrisko notāciju specifikācijām ir sekojoša divdaļīga struktūra:

- Signatūras kortežs $\langle S, \Omega \rangle$, kur S ir *sortu* kopa, bet Ω ir šo sortu operāciju kopa;

- Specifikācijas kortežs $\langle \Sigma, \Phi \rangle$, kur Σ ir signatūras algebras, bet Φ ir aksiomu kopa, kas apmierina šīs algebras.

Semantiski tas nozīmē, ka specifikācijas modelē kā algebras, un katra specifikācija definē savu algebru klasi. Pārsvarā algebriskās specifikācijas izmanto, lai aprakstītu moduļu ārējos interfeisus

2.3.2. Specifikāciju morfismi

Aplūkosim kategoriju **SPEC**, kurā objekti ir formālo specifikāciju moduļi, bet bultas nosaka, kā vienas specifikācijas vārdnīca tiek attēlota citas specifikācijas vārdnīcā, saglabājot tajā esošo teorēmu patiesumu.

Kategorijas **SPEC** morfisms sastāv no sekojošām divām daļām:

- *Signatūras morfisma* (t.i. vārdnīcas attēlojuma), kas nosaka kā vienas specifikācijas sorti un operācijas attēlojas citas specifikācijas sortos un morfismos, vienlaicīgi saglabājot katras operācijas rangu;
- *Specifikācijas morfisma*, kas specificē, kā katra vienas specifikācijas aksioma attēlojas citas specifikācijas teorēmā, saglabājot tās patiesumu.

Papildus tam, visu laiku jānodrošina arī pierādījumu patiesums, t.i. nepieciešams pārbaudīt katru teorēmās pārveidoto aksiomu. Šo uzdevumu automatizēšanu var uzdot teorēmu pierādīšanas rīkiem.

Šī principa galvenā priekšrocība ir tā, ka šādām kompozīcijām var pielietot kategoriju teorijas operācijas. Tiek izdalīti primitīvi horizontālās un vertikālās kompozīcijas operāciju veidi.

Horizontālās kompozīcijas primitīvi ir:

- *Translēšana*, kas kategoriju teorijas terminos ir izomorfiska kopēšana (t.i. vienkārša pārdēvēšana). Izmantojot šo operāciju, iespējams pārbaudīt, vai divas vai vairāk specifikācijas ir savstarpēji identiskas;
- *Importēšana*, kas ļauj vienas specifikācijas interfeisa daļu iekļaut citas specifikācijas interfeisā (tajā pat laikā ir pieļaujama arī translēšana). Šo operāciju var izmantot, lai specifikācijas papildinātu ar papildus servisiem;
- *Apvienošana*, kas kategoriju teorijas terminos ir līdzrobeža (t.i. specifikāciju kompozīcija). Izmantojot šo operāciju, var apvienot vairākas specifikācijas vienā, t.i. veikt šo specifikāciju integrāciju.

Papildus tam, var izmantot arī *vertikālās* kompozīcijas primitīvu — *uzlabošanu*. Tas ir attēlojums starp specifikāciju un tās realizāciju (vai realizācijām). Uzlabošanas operācija ļauj veikt detalizāciju, izvēlēties starp vairākiem projektējuma virzieniem, pievienot dažādus ierobežojumus, un pat dažādiem specifikāciju moduļiem pielietot dažādas specifikācijas valodas.

Parasti šādās metodēs tiek izmantots princips "pareizs pēc konstrukcijas", kas nozīmē, ja jebkurā izstrādes posmā sistēmas komponentes tiek uzskatītas par pilnīgām, lai, komponējot kopā, saglabātos visas īpašības. Tomēr prakse rāda, ka tas ne vienmēr ir iespējams. Proti, modelēšanas un specificēšanas procesi parasti vai nu satur, vai arī izraisa nekonsistenci, vai arī kļūdas cilvēcisko faktoru ietekmē.

2.3.3. Specifikāciju moduļu vispārināšana

Mūsdienu datorsistēmas ir pārāk sarežģītas, lai tās varētu radīt, uztvert un pārvaldīt kā vienu veselumu. Lai ar to cīnītos, programminženierija ir adoptējusi modulārās dekompozīcijas principu "skaldi un valdi", kas, sadalot sistēmu, relatīvi neatkarīgās daļās, ļauj to labāk uztvert. Tā rezultātā moduļa koncepts ir kļuvis par vienu no būtiskākajiem — tikai labi strukturētas sistēmas var veiksmīgi uzprojektēt, realizēt un uzturēt. Moduļa koncepts ir jāuztver ar ļoti plašu nozīmi, sākot ar klasiskā moduļa izpratni — funkciju bibliotēku, līdz pat objektu klases modulim, šablonam un komponentei Web servisā.

Arī šajā jomā KT var palīdzēt, piedāvājot universālu moduļa definīciju. Šāda moduļa abstrakcija nav piesaistīta specifiskam moduļu veidam, tajā pat laikā to var pielietot ļoti dažādiem moduļiem — gan funkcionālajiem, gan datu tipu, gan predikātu, gan stāvokļu mašīnu. Promocijas darbā katrs tiek aplūkots detalizētāk.

2.3.4. Viedokļu specifikāciju integrācija

Datorsistēmu izstrādes process ir efektīvāks, ja izstrādātāji var izlemt, kādas metodes un valodas labāk izmantot, gan vadoties pēc personīgās pieredzes ar noteiktām valodām un metodēm, gan pēc izstrādājamās datorsistēmas specifikas. Ja datorsistēmas izstrādē izmanto vairākas metodes un/vai valodas, tad neizbēgami rodas jautājumi par daļējo rezultātu kopējo integrāciju. Piemēram, ja tiek izmantotas vairākas formālās notācijas, ar katru specificējot noteiktu sistēmas pozīciju jeb viedokli, tad, lai iegūtu datorsistēmas kopējo specifikāciju, dažādo viedokļu specifikācijas nepieciešams apvienot vienā kopējā, kas definē visu datorsistēmu kopumā.

Šajā gadījumā, lai konstatētu, ka dažādu pozīciju prasību specifikācijas nav pretrunīgas, nepieciešams veikt specifiskas pārbaudes un saskaņošanu. Ja šādas pārbaudes ir automatizējamas, tad pretrunu meklēšanu un atklāšanu var realizēt rīkos, un analizēt to pārskatus.

Daļējo specifikāciju prakse rāda, ka specifikācijas ir vieglāk integrēt tad, ja tās ir aprakstītas izmantojot vienus un tos pašus terminus [17]. Šim gadījumam ir izstrādātas atbilstošas tehnikas daļējām homogēnām specifikācijām, t.i. specifikācijām ar kopēju notāciju.

Bet situācijās, kad ir dotas daļējas formālās specifikācijas, bet dažādās notācijā, piemēram, algebriskās — datu struktūru viedoklim un CCS (*Calculus of Communicating Systems*) [58] — procesu pozīcijai, tad nepieciešams veikt dažādo valodu integrāciju. Viens no šādas integrācijas karkasiem ir balstīts uz kategoriju teorijas izmantošanu [20].

Piemēram, sistēma tiek aplūkota no trīs viedokļiem. Pirmais — datu viedoklis apraksta sistēmu kā datu struktūru. Otrais viedoklis definē sistēmas operācijas kā abstraktus stāvokļus. Visbeidzot, trešais viedoklis — sistēmas uzvedības viedoklis, izmantojot tādus procesu algebras operatorus kā nedeterminēta izvēle, secīga un paralēla kompozīcija, apraksta sistēmas uzvedību, kurā par darbībām tiek izmantotas otrā viedokļa operācijas.

Ar kategoriju teorijas palīdzību šādu kombinēšanas mehānismu var vispārināt. Proti, tipus uztverot kā kategoriskas specifikācijas, kas, formējot kategoriju, ļauj aprakstīt elementus (modeļus). Līdz ar to jebkura formāla notācija, kas atļauj izveidot modeļu kategoriju, var tikt uzskatīta par specifisku stāvokļu valodu. Par stāvokļiem šādās valodās var būt kopas, algebras, relācijas vai grafi.

2.3.5. Kopsavilkums

Tika apskatītas iespējas formalizēt formālu notāciju integrāciju izmantojot KT principus. Šādu integrāciju nosaka nepieciešamība pēc pietiekoši universāliem formālismiem kas nav piesaistīti pašām integrējamajām notācijām. Savukārt formāla integrācija ļauj izmantot dažādas notācijas, kas ir paredzētas specifisku sistēmas viedokļu aplūkošanai.

Specifikācijas iekapsulēšana modulī tiek veikta, izmantojot divus pamatprincipus: signatūras un lokalitātes. Signatūra ir nepieciešama, lai izdalītu vieno specifikāciju no pārējām, savukārt lokalitātes princips nosaka, ka specifikācijas atribūtus var izmainīt tikai ar pašas specifikācijas operācijām. Iepriekš minēto īpašību realizēšanu un pierādīšanu var realizēt rīkos, kas automātiski izrēķina šķēršļus vai izgrūdumus — ja specifikācijas ir kategorijas objekti, tad līdzrobežas ir šo specifikāciju iespējamās integrācijas (kombinācijas), bet robežas — pārklājumi (kopīgās īpašības).

2.4. UML stāvokļu mašīnu modeļu formālās pārbaudes MDA kontekstā

Promocijas darba ceturtā nodaļa veltīta UML stāvokļu mašīnu modeļu pārbaudēm ar mērķi veikt formālas pārbaudes pirms datorsistēmas modeļi tiek transformēti programmēšanas valodu kodā.

Lai pārliecinātos par izstrādājamā modeļa pareizību, nepieciešams veikt papildus pārbaudes, kas ir augstākā abstrakcijas līmenī nekā no modeļa ģenerētais kods. Proti, nepieciešams veikt formālas *modeļu pārbaudes*, kuru mērķis ir noskaidrot, vai dotais modelis ir semantiski pilnīgs un nepretrunīgs. Šādas pārbaudes var veikt, piemēram, pielietojot dažādas formālas metodes.

Šādas pārbaudes ir būtiskas sevišķi datorsistēmu dinamisko jeb uzvedības modeļu gadījumā, kad nepieciešams pārliecināties, vai modelis pareizi atspoguļo datorsistēmas uzvedību.

Darbā tiek apskatītas četras formālās modeļu pārbaudes metodes, ar kuru palīdzību var pārbaudīt UML uzvedības diagrammu veida — stāvokļu mašīnu diagrammu modeļus. Katrai salīdzināmai metodei tiek dots tās apskats un analizētas priekšrocības un trūkumi [8, 3].

2.4.1. UML modeļu formālās pārbaudes metožu analīze

Kaut arī formālām metodēm tiek minēti arī negatīvi aspekti, tomēr tās var spēlēt būtisku lomu, lai risinātu grafisko notāciju formalizācijas problēmas. Formalizējot UML stāvokļu mašīnu diagrammas notāciju, var novērst virkni pretrunu un neprecizitāšu, kas rodas modeļus transformējot kodā. Vienlaicīgi tiek veicināta informācijas par starpmodeļu attiecībām uzturēšana, kas atvieglo koda ģenerēšanu un modeļu īpašību analīzi.

Darbā tiek apskatītas sekojošas četras metodes (nosaukumi ir darba autora doti un veidoti pēc šo metožu autoru uzvārdiem):

- Fernandesa-Tovala pārrakstīšanas loģikas metode [32];
- Cao un kolēģu grafu transformāciju metode [78];
- Knapa-Merca metode un rīks *Hugo/RT* [47];
- Sekerinska-Zuroba abstraktas mašīnas notācijas un B metode [68].

Apskatāmo metožu izvēle balstās uz iespējamo realizāciju daudzveidību, t.i. šīm metodēm kopīgs ir tikai tas, ka tās mēģina formalizēt UML stāvokļu mašīnu diagrammu notāciju un modeļus transformēt formālā notācijā, kurai pēc tam veic modeļu pārbaudes. Skatoties no realizācijas aspektiem, šīs metodes ir pilnīgi atšķirīgas.

Apskatīto metožu īpašību un iespēju kopsavilkums apkopots 1. tabulā, kuras pirmajā kolonnā ir uzskaitītas salīdzinošās pazīmes, bet pārējās kolonnas raksturo katras metodes iespējas/raksturojumus pēc šīm pazīmēm. Izvēlētās pazīmes raksturo galvenās iespējas, kuras būtu jānodrošina modeļu pārbaudes metodēm, kā arī metodēs izmantotās konceptuālās pieejas. Galvenie secinājumi ir sekojoši.

Visas aplūkotās metodes nodrošina stāvokļu mašīnas statiskās semantikas pārbaudi — tās ļauj iegūt atbildi uz jautājumu, vai modelis ir pabeigts — tam ir ieeja, sākuma stāvoklis, starpstāvokļi un beigu stāvoklis ar izeju, t.i. vai modelis ir sintaktiski pareizs, bet ne uz to, vai tas ir loģisks. Dinamiskās semantikas pārbaudi var veikt tikai tām metodēm, kas ļauj iegūt formālas specifikācijas, kas ir izpildāmas.

Dinamiskās semantikas pārbaudes mērķis ir noskaidrot, vai modeļa elementi ir savienoti korekti. Zināmā mērā šo problēmu var risināt, izvēloties atbilstošu formālo

1. tabula. UML stāvokļu mašīnu diagrammu formālas pārbaudes metožu salīdzinājums

	Fernandess-Tovals	**Cao**	**Knaps-Mercs**	**Sekerinskis-Zurobs**
Izmantotā pieeja / rīks	pārrakstīšanas loģika / *Maude*	grafu transformācijas un Petrī tīkli	galīgais automāts un temporālā loģika / *Hugo/RT*	AMN un B metode / *iState*
Statiskās semantikas atbalsts	jā	jā	jā	jā
Dinamiskās semantikas atbalsts	nē	jā	daļēji (transformējot Java kodā)	daļēji (transformējot kodā)
Modeļa izpildīšanas iespēja	tikai statiskās semantikas pārbaudei	jebkurš rīks kurā var izpildīt Petrī tīklu modeļus	jā (tikai prototipēšanas nolūkiem)	jā (kā gala programma)
Balstās uz UML metamodeli	jā	jā	jā	jā
Modeļa nepilnību automatizēta iegūšana	tikai statiskai semantikai	daļēja (jāizmanto Petrī tīklu atkļūšana)	tikai statiskai semantikai	tikai statiskai semantikai
Programmas koda ģenerēšana	nē	nē	jā (neoptimizēts Java kods)	jā

valodu, kā tas ir Cao metodes gadījumā, kur Petrī tīklu formalizācija un esošo rīku pieejamība ļauj veikt padziļinātas modeļa pārbaudes.

Kā zināms, modeļa izpildīšana ir iespējamo situāciju modelēšana stāvokļu telpā. Tā kā vispārīgā gadījumā stāvokļu telpa var būt neierobežoti liela, tad no modeļa izpildīšanas stratēģijas (sevišķi nedeterminētu un laiksakritīgu stāvokļu gadījumā) būs atkarīgs modeļa adekvātuma rezultāts.

Kopīgs visām apskatītajām metodēm ir tas, ka tās balstās uz atbilstošiem UML metamodeļiem, kas, mainoties UML versijām, ļauj veikt vienkāršākas metožu atbilstības uzlabošanas saskaņā ar UML metamodeli, papildus novēršot metamodeļa nepilnības formalizācijas aspektā, t.i. tās situācijas, kuras UML metamodelis nedefinē.

Lai arī cik formālas vai automatizējamas būtu modeļu pārbaudes, neizbēgami rodas situācijas, kurās lēmums ir jāpieņem modeļa izstrādātājam. Savukārt modeļa izstrādātāja lēmums ir atkarīgs no iegūtās informācijas par modeļa pārbaudes rezultātiem. Līdz ar to, pārbaudes rezultātu atbilstoša noformēšana ļauj pieņemt kvalitatīvākus lēmumus par modeļa turpmāko uzlabošanu. Visas aplūkotās metodes ļauj iegūt modeļa statiskās semantikas nepilnības, kas, faktiski, nozīmē, ka modeļa uzvedības (t.i. dinamikas) pārbaude un nepilnību novēršana balstās tikai uz modeļa izstrādātāja pieredzi un intuīciju.

Visbeidzot, modeļu vadāmas programmatūras izstrādes kontekstā, pilnvērtīga programmas koda ģenerēšana no modeļa, bez izstrādātāja līdzdalības, ir būtisks solis, kas ļauj gūt pārliecību, ka transformēšana uz kodu notiek pēc strikti definētiem likumiem, un ka iegūtā programmatūra atspoguļo modeli. Diemžēl, pašreiz šo aspektu pilnībā neatbalsta neviena no apskatītajām metodēm, kaut arī vairākas no tām to piedāvā risināt, tomēr, iegūtais rezultāts ir vai nu neefektīvs no izpildīšanas viedokļa (Knapa-Merca metode), vai arī definē ierobežojumus attiecībā pret modeli un programmēšanas valodām (kā Sekerinska-Zuroba metodē).

Aplūkoto metožu būtiskākā problēma ir jautājumā, līdz kādam līmenim formalizēt UML stāvokļu mašīnu modeļu operāciju semantiku (matemātisko pamatojumu interpretācijai un izpildīšanas soļu secībām), kuru UML projektētāji nav pilnībā

nodefinējuši. Līdz ar to pastāv iespēja, ka formalizācija var atšķirties starp dažādām metodēm, tādā veidā radot viena modeļa vairākas interpretācijas. Viens no risinājumiem varētu būt UML stāvokļu mašīnas transformēšana hierarhiska galīgā automāta formātā, kas ir pilnībā formāls. Diemžēl tas nenodrošinātu vairāku būtisku UML stāvokļu mašīnu īpašību attēlošanu, starp kurām ir aktivitātes, ieejas un izejas darbības, pabeigšanas notikumi un pārejas, vēstures un sazarošanās pseidostāvokļi; un to pievienošana nav triviāla.

2.4.2. Kopsavilkums

Šajā nodaļā tika dots tādu modeļu pārbaudes formālo metožu apskats, kurās tiek izmantotas transformācijas formālās specifikāciju valodās, ar mērķi izmantot formālo specifikāciju priekšrocības modeļu nepilnību atklāšanā un novēršanā. Tika ieskicētas būtiskākās problēmas, ar kurām jāsaskaras, kad vēlas veikt UML stāvokļu mašīnu diagrammu pārbaudes.

Šī tēma ir aktuāla arī modeļu vadāmas arhitektūras ietvarā, kurā tieši uz modeļiem gulstas primārā atbildība par sistēmas atbilstību prasībām, līdz ar to, jo efektīvāki būs rīki, kas spēs veikt modeļu pārbaudes augstā abstrakcijas līmenī, jo precīzāki modeļi tiks veidoti un būs iespējams ģenerēt efektīvāku programmas kodu. Šādi soļi ir sevišķi būtiski tieši datorsistēmas dinamisko jeb uzvedības modeļu gadījumā, kad nepieciešams pārliecināties, vai modelis pareizi atspoguļo datorsistēmas uzvedību.

Kopējais secinājums ir tāds, ka ir pieejamas dažādas pieejas diagrammu pārbaudēs, bet neviena no tām nepretendē uz vienīgo labāko — katrai metodei ir savas priekšrocības un trūkumi un/vai ierobežojumi. Tomēr vislielākā nepilnība, ar kuru jāsaskaras ikvienas metodes gadījumā ir tā, ka pašreizējā UML specifikācija nav pilnībā formalizēta, kā rezultātā metodēs tiek veikta pielāgota formalizācija, kas, bieži vien, tikai ar niansēm atšķiras starp dažādām metodēm.

2.5. UML un formālo notāciju integrācijas ietvars

Šī promocijas darba piektajā nodaļā tiek izklāstīts darba galvenais rezultāts — formālo notāciju un UML modeļu integrācijas ietvars [6, 7].

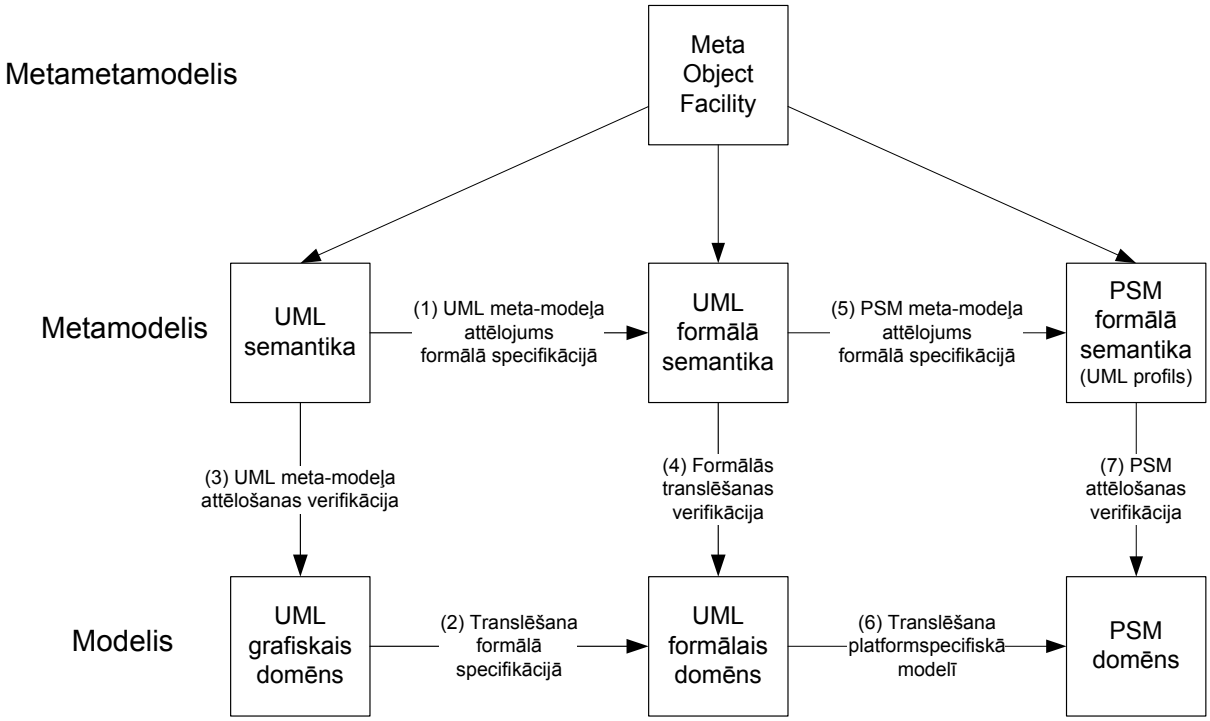
2.5.1. UML modeļu transformācijas

Apskatot UML modeļu transformācijas jautājumus, nepieciešams izvirzīt noteiktus nosacījumus un kritērijus, balstoties uz kuriem veikt transformāciju iespēju aprakstīšanu un praktisko pielietošanu.

Transformācijām MDA kontekstā nepieciešamas apmierināt vismaz sekojošas īpašības [46]: transformāciju likumu definēšana, transformāciju pielāgošana ar parametru palīdzību, inkrementālās konsistences nodrošināšana, divvirziena trasējamība, transformācijas objektu atbalsts, transformāciju klases un hierarhijas.

Kamēr, šādus nosacījumus apmierināt nebūs iespējams, tikmēr arī transformāciju lietderība netiks pilnībā izmantota, un pēc darba autora domām, MDA vīzija nebūs pilnībā īstenota.

Kā viens no iespējamiem šīs problēmas risinājumiem ir pielietot kategoriju teorijas principus, jo ar KT konstrukcijām var formāli aprakstīt katru uzlabošanas un transformācijas soli, katra modeļa specifikācijai.



2. att. UML modeļu transformēšanas un verificēšanas ietvars

Galvenais uzsvars piedāvātajā metodē tiek likts uz KT praktisko pielietojumu, mazāk uz teorētiskiem aspektiem. Proti, izmantojot formālo specifikāciju valodu pētījumus no kategoriju teorijas viedokļa, tiek pielietotas kategoriju teorijas konstrukcijas specifikāciju transformācijās un transformāciju semantikas analīzē.

2.5.2. Formālo specifikāciju integrācija MDA ietvarā

Būtiskākais modeļu vadāmā arhitektūrā integrējamo formālismu nosacījums ir, lai UML un UML lietojumu specifikāciju formalizāciju varētu realizēt viegli paplašināmā un automatizējamā veidā, un lai integrācija atbalstītu specifikāciju kompozīciju, kā arī rīku savstarpējo sadarbību. Līdz ar to promocijas darbā piedāvātajā ietvarā, uzsvars tiek likts uz metamodelēšanu un tās verificēšanas procesu, kā arī unificējamu rīku atbalstu, kas ļauj veikt pusformālu modelēšanas notāciju (piemēram, UML) formalizāciju.

Cits būtisks aspekts ir, lai ietvars pamatotos uz MOF principiem, t.i. lai tiktu ņemts vērā UML metamodelis, kas, līdz ar to, arī ir jāformalizē. Piedāvātais ietvars šo aspektu ņem vērā.

Šajā darbā piedāvātā UML formalizācijas procesa integrācijas ietvara vispārēja shēma parādīta 2. attēlā. Tā kā tas ir pietiekoši abstrakts, to var pielietot jebkurai no darbā apskatītajām formālo specifikāciju valodām.

KT pielietojums izpaužas tajā aspektā, ka galvenais uzsvars tiek likts uz attiecībām starp ietvara virsotņu elementiem, proti, bultām. Respektīvi, ja katra virsotne ir formalizēta kategorijā, tad bultas atspoguļo attiecības starp šīm kategorijām, respektīvi — funktorus. Šī ietvara būtiskākā atšķirība no citiem formalizācijas ietvariem ir tā, ka ar formālām specifikācijām var aprakstīt objektu struktūru un uzvedību, kamēr uz KT bāzētā pieeja tiešā veidā parāda attiecības starp specificējamiem objektiem, respektīvi, uzsvars tiek likt uz transformācijām.

Piedāvātajā integrācijas ietvarā bultu (funktoru) nozīme ir sekojoša. Attēlojums (1) apraksta UML metamodeļa formalizāciju. Ar formālu likumu palīdzību tiek aprakstīts,

kā transformēt UML semantikas metaobjektus formālā specifikācijā (piem., Z, Maude vai TLA$^+$). Papildus tiek aprakstīta katra transformēšanas likuma nozīme un iespējamās transformēšanas likumu alternatīvas.

Attēlojums (2) specificē programmatūras un rīku atbalsta nepieciešamību, lai automātiski transformētu UML modeļus formālajās specifikācijās, pēc (1) attēlojumā definētiem formalizācijas likumiem. Rezultāts ir formāla specifikācija jeb UML formālais domēns.

Attēlojums (3) specificē visas iespējamās UML diagrammas, kuras var izveidot no UML metamodeļa ar rīka palīdzību. Citiem vārdiem sakot, tā ir UML diagrammu interpretācija UML metamodelī. Attēlojumi (1) un (3) kopā veido pamatu UML diagrammu turpmākai formalizācijai.

Attēlojums (4) nodrošina UML diagrammu translēšanas verifikāciju. Ar tā palīdzību notiek pārliecināšanās, vai no UML grafiskā domēna iegūtās formālās specifikācijas atbilst abstraktajai UML teorijai jeb formālajai UML semantikai.

Attēlojums (5) ir attēlojums uz platformspecifisku (PSM) metamodeli, kurš, piemēram, varētu būt kāds no UML profiliem vai transformēšanas modeļiem. Šis attēlojums tiek aprakstīts ar formālās specifikācijas valodas palīdzību. Zinot specifikāciju attēlojumu semantiku, un izmantojot atbilstošus transformēšanas likumus, ir iespējams formālo specifikāciju attēlot ļoti precīzi (attēlojums (6)), un pēc tam šo transformāciju verificēt ar attēlojuma (7) palīdzību, atbilstoši platformspecifiskā modeļa formālās semantikas prasībām.

Šajā ietvarā, katra specifikācija tiek aplūkota kā neatkarīgs objekts, savukārt morfismi starp šiem objektiem parāda specifikāciju savstarpējo saistību. Tas ļauj specificēt datorsistēmas noteiktus aspektus tajās specifikāciju valodās, ar kurām šos aspektus un īpašības ir visefektīvāk izteikt (t.i. kuras satur šādu īpašību aprakstīšanas iespējas). Piemēram, ar vienu specifikāciju apraksta stāvokļus, kamēr ar citu — datu struktūras un/vai procesus.

KT pieejas gadījumā, UML diagrammas automātiskās transformācijas formālajā specifikācijā rezultāta pareizība reducējas uz pierādījumu, ka transformatora ģenerēto modeļu klase iekļaujas UML metamodeļa modeļu klasē. Respektīvi, nepieciešams pierādīt, ka eksistē morfisms no katras UML metamodeļa teorijas uz noteiktas UML diagrammas attēlojuma teoriju.

Promocijas darbā metamodeļu līmenī ir definēts UML stāvokļu mašīnu integrācija ar Z notāciju un Maude algebrisko pieeju.

Maude specifikācija tiek sasaistīta ar UML pakotni, bet katrs no Maude specifikācijas objektu moduļiem, tiek sasaistīts ar UML klašu diagrammas klasi.

Z specifikācijai atbilst viena stāvokļu mašīnas diagramma, bet katram Z specifikācijas paragrāfam atbilst UML stāvoklis. Respektīvi ar Z specifikāciju var specificēt, stāvokļu mašīnas invariantu, kurš ir spēkā, neatkarīgi no stāvokļa, kā arī katram stāvoklim papildus var specificēt papildus specifikāciju Z notācijā, kas var tikt izteikta vai nu kā shēma, aksioma, vai arī vispārīgā Z notācijas pierakstā.

Šo modeļu attēlošanas verifikāciju var automatizēt. Proti, ja ir formalizēts gan metamodelis, gan arī modelis, tad tā attēlošanas verifikācijas procesa pārbaude reducējas uz metamodeļa aksiomu transformāciju modeļa teorēmās — vai modeļa atspoguļojums nav pretrunā ar metamodelī definētām aksiomām. Ja tiek atklātas pretrunas, tās var uzreiz novērst, jo var noteikt, tieši kurš modeļa elements ir nekonsistents.

Modeļa transformēšana PSM ir nākamais solis pēc tam, kad esam pārliecinājušies, ka PIM ir korekts no UML formālās sintakses un semantikas viedokļa. PSM modeļa

galvenais mērķis ir, lai no tā varētu automātiski transformēt izpildāmu aplikācijas kodu. Līdz ar to šajā gadījumā nav tik būtiski noskaidrot, vai modelis ir semantiski pareizs (ja tiek pieņemts, kas tā pareizība ir pārbaudīta jau PIM), cik tas ir determinēts un viennozīmīgi realizējams. Proti, nepieciešams formalizēt katru modeļa elementu līdz tādam līmenim, lai no tā varētu ģenerēt izpildāmu specifikāciju vai kodu. Šajā gadījumā, katram platformspecifiskā modeļa UML profilam nepieciešams formalizēt un parādīt tā konceptu attiecības formālā specifikācijā.

2.5.3. Kopsavilkums

Piedāvāto ietvaru raksturo princips, ka katru UML modeli un tā formalizēto specifikāciju aplūko kā neatkarīgu objektu, savukārt morfismi starp tām specificē, kā šie objekti ir savstarpēji saistīti. Tas ļauj specificēt noteiktus datorsistēmas aspektus tādā notācijā, kas ir vislabāk piemērota aplūkoto datorsistēmas īpašību aprakstam. Piemēram, viena specifikācija apskata datorsistēmas procesus, kamēr citas — datu struktūras un stāvokļus.

Ietvars ļauj integrēt arī daļēju specifikāciju realizācijas, kurā katrs modelis tiek specificēts un pārbaudīts dažādās specifikāciju valodās, kas ir vispiemērotākās dotajam uzdevumam, un pēc tam, izmantojot specifikāciju integrācijas mehānismus, individuālās modeļa specifikācijas tiek apvienotas, veidojot datorsistēmas kopējo modeli.

Savukārt kategoriju teorijas pieeja nodrošina labi saskaņotu un pamatotu teorētisko bāzi formālo specifikāciju valodu struktūru attēlošanai un sasaistīšanai ar UML metamodeli.

2.6. Ietvara pielietošanas demonstrējums — datorsistēma *TraSer*

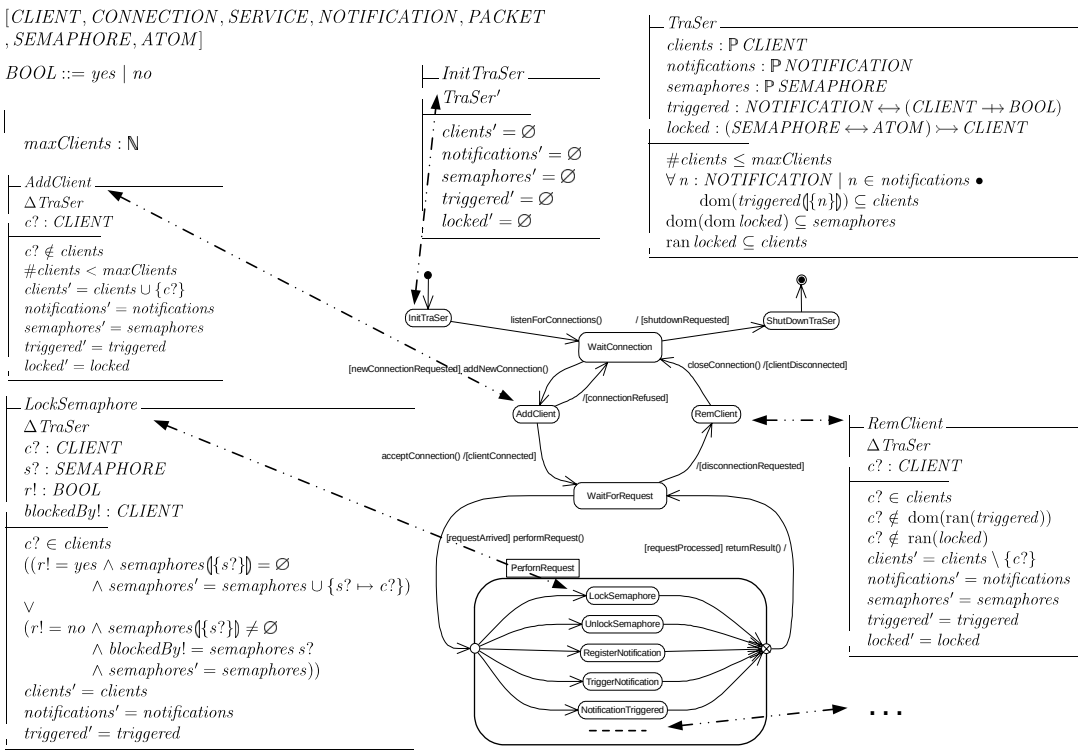
Promocijas darba sestajā nodaļā, ar praktiski realizētu problēmvides piemēra palīdzību, tiek demonstrēta promocijas darbā piedāvātā modeļu integrācijas ietvara pielietošana.

Piemēra problēmvide ir saistīta ar izkliedētu transakciju atbalsta realizāciju. Darbā tiek detalizēti aprakstīta šīs datorsistēmas *TraSer* (*Transakciju Serviss*) projektēšana gan UML notācijā, gan formālajās specifikācijās.

Piedāvāto risinājumu var pielietot tādu datorsistēmu atbalstam, kurās ar datu bāzes vadības sistēmas (DBVS) piedāvātajām iespējām datu bloķēšanas (*locking*) nodrošināšanai nav pietiekoši, lai efektīvi varētu realizēt klientu aplikācijas, kas darbojas biznesa procesu līmenī.

Integritātes nodrošināšanai realizēts semaforu mehānisms, kas nav atkarīgs no izmantojamās DBVS un tās iespējām transakciju nodrošināšanai. Vēl vairāk, tajā realizēti papildus servisi, kas tiešā veidā nav saistīti ar transakcijām un integritātes nodrošināšanu. Piemēram, sistēma *TraSer* paredz notifikāciju atbalsta mehānismus, kad aktīvās klienta aplikācijas tiek informētas par kādu globālu datu (piemēram, biznesa procesa parametru) aktualizēšanas nepieciešamību. Respektīvi, lai klienta aplikācijai nebūtu regulāri jāpārbauda kādu nosacījumu izmaiņa, tā var "pasūtīt" dotā nosacījuma izmaiņas notifikāciju, un pie šo nosacījumu izmaiņas stāšanās spēkā, automātiski saņemt ziņojumu, un atbilstoši reaģēt.

Vispārinot problēmu, šāda tipa uzdevumus var pielietot ne tikai DBVS, bet arī tur, kur ir nepieciešami izkliedēti sinhronizēšanas un bloķēšanas mehānismi plašākā



3. att. Sistēmas *TraSer* UML stāvokļu mašīnas un Z shēmu specifikāciju integrācijas pārskats

aspektā. Bet tā kā *TraSer* ir koordinējoša sistēma, kas nodarbojas ar izkliedētu klienta aplikāciju sinhronizēšanu, tad tai izvirzās atbilstoši kvalitātes nosacījumi — *TraSer* atteices gadījumā var pārtraukt darboties arī visas klienta aplikācijas, kas nav pieļaujams. Līdz ar to šāda tipa sistēmai formālo metožu pielietošana ir aktuāla.

Visas promocijas darbā dotās *TraSer* UML modeļu diagrammas attēlojas 2. attēlā dotā integrācijas ietvara virsotnē 'UML grafiskais domēns'.

TraSer Maude algebriskās specifikācijas tika pārbaudītas arī rīkā *Maude*. Pārbaudes rezultātā specifikāciju sākotnējās versijās atklājās sintakses kļūdas un nepilnības, kas tika novērstas. Šādu specifikāciju automātiska verifikācija ļauj iegūt pārliecību, ka UML modeļus iespējams transformēt atbilstošās formālās notācijās, kuras var pēc tam pārbaudīt ar šo notāciju atbalstošajiem rīkiem, ļaujot novērst nepilnības sākotnējos UML modeļos.

Translācija no UML klašu diagrammas un stāvokļu mašīnas uz Z specifikācijām tiek veikta ar mērķi formāli specificēt sistēmas uzvedību, kā modeli, kas maina savu stāvokli pēc noteiktiem nosacījumiem.

Piesaistot katru promocijas darbā definēto *TraSer*Z shēmu atbilstošam sistēmas *TraSer* stāvokļa mašīnas stāvoklim, tiek iegūta bagātinātāka specifikācija, nekā tas būtu, ja atsevišķi tiktu izmantotas Z notācijas specifikācijas un UML stāvokļu mašīna (3. attēls).

No vienas puses, ar UML stāvokļu mašīnu tiek definēts viegli uztverams, bet neformāls sistēmas dinamikas aspekts, bet no otras puses, ar sākotnēji grūtāk uztveramas Z notācijas palīdzību, modelis tiek papildināts, realizējot sistēmas modeļa formalizāciju. Piemēram, ar Z notāciju tiek specificētas sistēmas īpašības, kuras tiešā veidā ar UML stāvokļu mašīnā nav parādītas. Tā Z shēma *TraSer* specificē sistēmas stāvokļa shēmu — t.i. sistēmas invariantu, kam ir jābūt apmierinātam, neatkarīgi no tekošā stāvokļa UML stāvokļu mašīnā.

Katra Z shēma, kas piesaistīta noteiktam UML stāvokļu mašīnas stāvoklim, formālā veidā specificē informāciju, tieši kādā veidā sistēmas stāvoklis izmainās.

Balstoties uz šādi integrētām specifikācijām, tiek ģenerēts platformspecifiskais modelis, kurā tiek ņemta vērā visu iesaistīto UML diagrammu un formālo specifikāciju informācija. Līdz ar to tiek iegūts bagātināts sistēmas modelis, no kura var ģenerēt detalizētāku aplikācijas kodu. Šis modelis atbilst 2. attēlā dotā integrācijas ietvara virsotnei 'PSM domēns'.

2.6.1. Kopsavilkums

Ar praktiski realizētas sistēmas *TraSer* piemēra palīdzību, tiek demonstrēti promocijas darbā piedāvātā integrācijas ietvara pielietošanas aspekti. Tiek definētas sistēmas funkcionālās prasības, demonstrēta sistēmas klašu un stāvokļu mašīnas diagrammas. Balstoties uz šīm diagrammām, tās tika translētas formālajās notācijās, kuras teorētiskie principi ir aplūkoti darba 5. nodaļā.

Aplūkotās aplikācijas problēmsfēra ir specifiska tajā aspektā, ka tai tiek izvirzītas noteiktas kvalitātes prasības. No vienas puses, tās darbības efektivitāte nav tik kritiska, lai realizēšanas obligāta prasība būtu formālo metožu izmantošana, tomēr, tās nekorekta darbība var novest pie nopietniem datu integritātes zaudējumiem. Līdz ar to ir nepieciešams izvirzīt atbilstošas kvalitātes prasības.

Šajā nodaļā demonstrētajā piemērā ir izmantots UML modeļu un formālo notāciju integrācijas ietvars. Tā kā UML notācija nav pilnībā formalizēta, bet ir grafiska, tad ar UML modeļiem tiek specificēti ērti uztverami projektējamās sistēmas artefakti. Savukārt formalizētas notācijas palīdz novērst UML modeļu nepilnības, tos papildinot ar formālām specifikācijām, bet formālo specifikāciju verificēšana rīkos, dod papildus pārliecību par specifikāciju pareizību.

Abu veidu notāciju apvienojums ļauj iegūt bagātinātāku platformspecifisku modeli, no kura, līdz ar to, var ģenerēt detalizētāku kodu. Bet, pielietojot pakāpenisku uzlabošanu, var detalizēti specificēt sistēmas uzvedību, lai tiktu ģenerēts pilnībā pabeigts aplikācijas kods, par kura pareizību var pārliecināties izmantojot transformācijās saglabāto trasējamības informāciju.

Piedāvātā integrācijas ietvara pielietošanas priekšrocība ir formāla pamatojuma izmantošana. Tomēr lai pilnībā izmantotu piedāvātā ietvara priekšrocības, nepieciešams realizēt arī atbilstošu rīku atbalstu, kas ir ārpus šī promocijas darba uzdevumiem.

DARBA REZULTĀTI UN SECINĀJUMI

Datorsistēmu izstrādes formālās metodes ir pietiekoši plaši izpētītas un apgūtas, un ir iegūta pietiekama teorētiskā bāze tam, lai aprakstītu to, ko datorsistēmas dara, kā dara un kāpēc dara. Vidēja apjoma kritisko lietojumu programmu konstruēšanai, ir izstrādātas un pielietotas semantisko teoriju, specifikāciju valodu, projektēšanas un verificēšanas metodes un rīki.

Aktuālākais izaicinājums ir formālo metožu integrācija tajos programminženierijas izstrādes procesos, kuros datorsistēmu korektai konstruēšanai un uzturēšanai, tās pašlaik netiek pilnībā efektīvi izmantotas. Izpētot formālo metožu un teoriju integrācijas iespējas ar daļēji formalizētām pieejām, var uzlabot pašreizējās programmatūras izstrādes industrijas prasmes un metodes.

Līdz ar to promocijas darba pētījumu galvenais mērķis bija piedāvāt matemātiski pamatotu formālismu, kas bāzēts uz kategoriju teoriju, un to pielietot UML modeļu semantikas formalizēšanā un atbilstoša integrācijas ietvara izstrādē, kas, modeļu vadāmas arhitektūrā, ļauj integrēt formālo notāciju semantiku ar UML modeļu semantiku.

Promocijas darba **galvenais rezultāts** ir izstrādātais un piedāvātais formālo specifikāciju valodu un UML modeļu integrācijas ietvars, kas pielāgots modeļu vadāmai arhitektūrai, par integrācijas formālais pamatojums izmantojot kategoriju teorijas morfismu un funktoru jēdzienu pielietojumu.

Saistība ar kategoriju teoriju izpaužas tajā aspektā, ka galvenais uzsvars tiek likts uz attiecībām starp ietvara virsotņu elementiem, proti, bultām. Respektīvi, ja katra virsotne ir formalizēta kategorijā, tad bultas atspoguļo attiecības starp šīm kategorijām — funktorus. Būtiskākā atšķirība no citiem formalizācijas ietvariem ir tā, ka ar formālām specifikācijām var aprakstīt objektu struktūru un uzvedību, kamēr uz kategoriju teorijas balstītā pieeja tiešā veidā parāda attiecības starp specificējamiem objektiem, respektīvi, uzsvars tiek likts uz transformācijām.

Risinājuma oriģinalitāte pamatojas uz trīs tehnoloģiju (formālo notāciju, UML un kategoriju teorijas) apvienošanu, lai integrētu modeļu vadāmā arhitektūrā. Risinājums veicina formalizācijas ieviešanu tajās sfērās, kuras nav kritiskas, bet, ja tajās pielieto formalizētas metodes, var uzlabot šādu sistēmu programmatūras kvalitāti, lielāko daļu nepilnību atklājot un novēršot jau projektēšanas posmos.

Tāpat tika izpētītas transformāciju iespējas un problēmas MDA kontekstā, un secināts, ka galvenais esošo modeļu vadāmo pieeju trūkums ir nepietiekošs matemātiskais pamatojums, bet pieejas, kas izmanto matemātisku pamatojumu nav populāras industrijā. Līdz ar to piedāvātais ietvars sniedz zināmu šīs problēmas risinājumu, un ir pielietojams ne tikai kritisku sistēmu projektēšanā, bet plašākās problēmsfērās, kur formālo metožu pielietošana nav pietiekoši izplatīta, tomēr programmatūras atteices nav vēlamas, jo var radīt būtiskus finansiālos zaudējumus.

Piedāvātais ietvars ļauj integrēt nepilnīgi formalizētas grafiskas notācijas ar pilnībā formalizētām pieejām, veicinot sistēmas specifikāciju viennozīmību un matemātisko pamatotību, kā arī veicinot formālu metožu izmantošanu programmatūras izstrādē. Kā viens no būtiskākajiem šķēršļiem formālo metožu plašākai izmantošanai tiek minēts to apgūšanas laiks un to atbalstošo rīku trūkums. Apgūšanas laiku var samazināt, izstrādājos rīkus, kuri ļauj formāli specificēt projektējamo sistēmu, bet vienlaicīgi nenovērš izstrādātāja uzmanību no pārlieku formālām detaļām un darbībām, kas ir pilnībā automatizējamas, piemēram, specifikāciju sintakses pārbaudēm un teorēmu pierādīšanām.

Piedāvātais ietvars ļauj saglabāt specifikāciju konsistenci un modularitāti, un ir

automatizējams rīkos, kuros efektīvāk ir manipulēt tieši ar simboliska rakstura informāciju (formālām specifikācijām) nekā ar grafisku (UML modeļiem).

Pateicoties kategoriju teorijas vispārināšanas un abstrakcijas iespējām, piedāvātais ietvars ļauj integrēt jebkuru formālu notāciju, neatkarīgi no tās satura. Ir tikai jāizveido šīs formālās notācijas metamodelis, un tas jāsasaista ar UML metamodeli.

Piedāvātais ietvars tika praktiski aprobēts ar piemēra palīdzību, realizējot datorsistēmas *TraSer* projektējumu. Šī sistēma ļauj risināt izkliedētu transakciju atbalsta uzdevumus, sinhronizējot pieeju objektiem starp klienta aplikācijām, kas izmantot šīs sistēmas pakalpojumus.

Visiem sākotnēji izvirzītajiem uzdevumiem ir rasti risinājumi:

- Izmantot kategoriju teorijas vispārināšanas koncepcijas formālo specifikāciju valodu integrēšanai UML modeļos, prasību specificēšanas un datorsistēmas projektēšanas posmos — izmantojot kategoriju teorijas morfismus un funktorus tika izveidots integrācijas ietvars, kurā galvenais uzsvars tiek likts uz attiecībām starp ietvara virsotnēm.
- Izpētīt daļējo specifikāciju integrācijas iespējas kopējā datorsistēmas specifikācijā, izmantojot kategoriju teorijas principus modeļu formalizēšanai ar formālo notāciju palīdzību — piedāvātajā integrācijas ietvarā var savstarpēji integrēt vairāk kā vienu formālo notāciju, kā rezultātā ar katru no tām tiek formāli specificēts noteikts sistēmas aspekts, bet visas kopā ar UML modeļiem veido daudzpusīgu sistēmas modeli.
- Aprobēt piedāvātos risinājumus UML stāvokļu mašīnu modeļu gadījumā, un pielietot praktiska uzdevuma atrisināšanai — ietvara pielietojums ir nodemonstrēts sistēmas *TraSer* projektējumā.
- Izstrādāt ieteikumus modeļu pārbaužu veikšanai ar UML stāvokļu mašīnu modeļiem — apkopotie secinājumi UML stāvokļu mašīnu modeļu pārbaudēm ļauj gūt priekšstatu par šādu metožu iespējām un trūkumiem, galvenais ko kuriem ir UML stāvokļu mašīnu nepilnīgā operāciju semantika.

Tomēr tie nav galīgi, un vēl ir uzlabojami vai papildināmi, piemēram, piedāvātajā ietvarā integrējot citas formālās notācijas, un ar neapskatītajiem UML modeļu veidiem.

Šī darba **turpmākie pētījumu virzieni** ir saistīti ar atbilstoša rīku atbalsta izstrādi, vai arī esošo modeļu vadāmo rīku paplašināšanu, realizējot tikai sasaistošo moduli, jo gan MDA rīki, gan atbilstošo formālo notāciju rīki jau ir pieejami.

Promocijas darbs apliecina, ka var veiksmīgi apvienot gan ar UML saistītās modeļu vadāmās arhitektūras, gan formālās metodes, demonstrējot formālo pieeju lietderību, ne tikai kritiskajām datorsistēmām, bet arī plašākās IKT sfērās, kurās formālo metožu pielietošana pašlaik nav pietiekoši izplatīta.

IZMANTOTĀ LITERATŪRA

[1] Abrial J.-R. The B Book: Assigning Programs to Meaning. Cambridge University Press, 1996.

[2] Alksnis G., Osis J. Kategoriju teorija datorzinātnēs// Scientific Proceedings of Riga Technical University. Computer Science, Applied Computer Systems, Vol.8, Nr.5 - Riga: RTU Publishing, 2001. - pp. 59–67.

[3] Alksnis G. UML stāvokļu mašīnu modeļu formālās pārbaudes metožu salīdzinoša analīze// Scientific Proceedings of Riga Technical University. Computer Science, Applied Computer Systems, Vol.26, Nr.5 - Riga: RTU Publishing, 2007. - pp. 28–37.

[4] Alksnis G. Category Theory in Software Engineering// Proceedings of the 5th International Baltic Conference (DB&IS 2002), Vol.1 - Tallinn: Institute of Cybernetics at TTU, 2002. - pp. 269–278.

[5] Alksnis G. Formal Specification from Category Theory Viewpoint// Scientific Proceedings of Riga Technical University. Computer Science, Applied Computer Systems, Vol.17, Nr.5 - Riga: RTU Publishing, 2003. - pp. 137–144.

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

[7] Alksnis G. Formal Methods and Model Transformation Framework for MDA// Proceedings of the 1st International Workshop on Formal Models (WFM'06). Eds. D. Kolář and A. Meduna. - Ostrava: MARQ, 2006. - pp. 87–94.

[8] Alksnis G. The Analysis of UML State Machine Formal Checking Methods// Proceedings of the 10th International Conference Information System Implementation and Modeling (ISIM'07). Eds: A. Kelemenová, D. Kolář, etc. - Opava: Silesian University, 2007. - pp. 131–136.

[9] Alksnis G., Asnina E., Osis J., Silins J. Formalization of Software Development: Problems and Solutions// Scientific Proceedings of Riga Technical University. Computer Science, Applied Computer Systems, Vol.22, Nr.5 - Riga: RTU Publishing, 2005. - pp. 204–216.

[10] Alksnis G., Osis J. Formalization of software engineering by means of the theory of categories// Scientific Proceedings of Riga Technical University, Computer Science, Applied Computer Systems, Vol.13, Nr.5 - Riga: RTU Publishing, 2002. - pp. 157–163.

[11] Asnina E. Formalization of Problem Domain Modeling within Model Driven Architecture. PhD thesis. Riga Technical University, 2006.

[12] Asņina Ē., Osis J. Programmatūras izstrādes formālās modelēšanas problēmas un perspektīvas// Scientific Proceedings of Riga Technical University, Computer Science, Applied Computer Systems, Vol.13, Nr.5 - Riga: RTU Publishing, 2002. - pp. 145–156.

[13] Asperti A., Longo G. Categories, Types and Structures — An Introduction to Category Theory for the Working Computer Scientist, Foundations of Computing Series, MIT Press, 1991.// Internet: `ftp://ftp.ens.fr/pub/dmi/users/longo/CategTypesStructures`

[14] Barendregt P. H. The Lambda Calculus: Its Syntax and Semantics. North Holland: Amsterdam, 1984.// Internet: http://www.andrew.cmu.edu/~cebrown/notes/barendregt.html

[15] Barr M., Wells Ch. Category Theory for Computing Science, 3rd Ed., Montreal: Les Publications CRM, 1999. - 526 p.

[16] Boiten E.A., Bujorianu M.C. Exploring UML Refinement through Unification// Workshop on Critical Systems Development with UML, <<U M L>> 2003.

[17] Boiten E.A., Derrick J., Bowman H., Steen M.W.A. Constructive consistency checking for partial specification in Z// Science of Computer Programming, 35(1):29-75, 1999.

[18] Bowen J. Formal Specification and Documentation using Z: A Case Study Approach, Revised 2003.// Internet: http://ftp.museophile.lsbu.ac.uk/pub/jpb/zbook.pdf

[19] Bowman H., et. al. A formal framework for viewpoint consistency// Formal Methods in System Design, 21:111-166, 2002.

[20] Bujorianu M. C. Integration of Specification Languages Using Viewpoints// Integrated Formal Methods, 4th International Conference, IFM 2004, Lecture Notes in Computer Science, Springer-Verlag, 2004. - pp. 421–440.

[21] Burstall R., Goguen J. The Semantics of CLEAR, a Specification Language// In: Proc. Advanced Course on Abstract Software Specification, LNCS, vol.86, Berlin: Springer, 1980. - pp. 292–332.

[22] Burstall R., Goguen J. Institutions: Abstract Model Theory for Specification and Programming. Journal ACM 39(1), 1992. - pp. 95–146. // Internet: http://doi.acm.org/10.1145/147508.147524

[23] Clavel M., Duran F., Eker S., et. al. Maude Manual. Version 2.1.1, 2005. // Internet: http://maude.cs.uiuc.edu

[24] Diskin Z., Kadish B. The Arrow Manifesto: Towards software engineering based on comprehensible yet rigorous graphical specification // Internet: http://citeseer.nj.nec.com/167037.html

[25] Duke R., Rose G. Formal Object-Oriented Specification Using Object-Z. Macmillan Press, 2000.

[26] Easterbrook S. Category Theory for Beginners // Internet: http://www.cs.toronto.edu/~sme/presentations/cat101.pdf

[27] Ehrich H., Sernadas A., Sernadas C. Objects, Object Types and Object Identification// In: H.Ehrig, et.al. (Eds.), Categorical Methods in Computer Science with Aspects from Topology, LNCS, vol. 393, Berlin: Springer, 1987. - pp. 142–156.

[28] Ehrig H., Fey W., Hansen H., Löwe M., Parisi-Presicce F. Categories for the Development of Algebraic Module Specifications// In: H.Ehrig, et.al. (Eds.), Categorical Methods in Computer Science with Aspects from Topology, LNCS, vol. 393, Berlin: Springer, 1987. - pp. 157–184.

[29] Eilenberg S., Mac Lane S. Group Extensions and Homology// Annals of Mathematics. Vol. 43, 1942. - pp. 757–831.

[30] Eilenberg S., Mac Lane S. General theory of natural equivalences. Transactions of the AMS, No. 58, 1945. - pp. 231–291.

[31] Fiadeiro J. L. Categories for Software Engineering. Springer-Verlag, 2004.

[32] Fernandez J. L., Toval A. Can Intuition Become Rigorous? Foundations for UML Model Verification Tools// In: Proceedings of the 11th International Symposium on Software Reliability Engineering (ISSRE 2000), IEEE Press, 2000. - pp. 344–355.

[33] Fokkinga M. A Gentle Introduction to Category Theory - the calculational approach, Version of June 6, 1994. // Internet: `http://wwwhome.cs.utwente.nl/~fokkinga/mmf92b.html`

[34] Goguen A. J. A Categorical Manifesto// In: Mathematical Structures in Computer Science, Vol.1, Nr.1, March 1991. - pp. 49–67.

[35] Goguen J. Parametrised Programming and Software Architecture// In: Proceedings of 4th International Conference on Software Reuse, IEEE CS Press, Silver Spring, 1996. - pp. 2–11.

[36] Goguen J. Reusing and Interconnecting Software Components// In: IEEE Computer, 19(2), 1986. - pp. 16–28.

[37] Goguen J. What is unification? A categorical view of substitution, equation and solution. Nivat, M., Aït-Kaci, H. (eds), Resolution of equations in algebraic structures. - Academic Press, 1989. - pp. 217–261.

[38] Goguen J., Thatcher J., Wagner E. An initial algebra approach to the specification, correctness and implementation of abstract data types// In: R.Yen (ed), Current Trends in Programming Methodogy IV. - London: Prentice Hall, 1978. - pp. 80–149.

[39] Goguen J., Winkler T., Meseguer J., et al. Introducing OBJ — Applications of Algebraic Specification using OBJ, 1993 // Internet: `http://www-cse.ucsd.edu/users/goguen/ps/iobj.ps.gz`

[40] Grothendieck A. Sur Quelques Points d'Algebre Homologique// Tehoku Mathematics Journal. Vol. 9, 1957. - pp. 119–221.

[41] Harel D. On Visual Formalisms// In: Communications of the ACM, volume 31, number 5, May 1988. - pp. 514–530.// Internet: `http://doi.acm.org/10.1145/42411.42414`

[42] Hoare C. A. R. Communicating Sequental Processes. Prentice-Hall, Englewood Cliffs, 1985.

[43] Holzmann G. J. The SPIN Model Checker: Primer and Reference Manual, Addison-Wesley, 2004.

[44] Jones C. B. Systematic Software Development Using VDM. Prentice-Hall, 1990.

[45] Kestrel Institute Home Page // Internet: `http://www.kestrel.edu`

[46] Kleppe A., Warmer J., Bast W. MDA Explained: The Model Driven Architecture: Pratice and Promise. Addison Wesley, 2003.

[47] Knapp A., Merz S. Model checking and code generation for UML state machines and collaborations// In: D. Haneberg, G. Schellhorn, and W. Reif, editors, 5th Workshop on Tools for System Design and Verification, Technical Report 2002-11, Institut fur Informatik, Universitat Augsburg, 2002. - pp. 59–64.

[48] Lamport L. Specifying systems: the TLA+ language and tools for hardware and software engineers, Addison-Wesley, 2002.

[49] Lamport L., Yu Y. TLC — The TLA+ Model Checker, Microsoft Research, 2003// Internet: http://research.microsoft.com/users/lamport/tla/tlc.html

[50] Larsen K. G., Peterson P., Yi W. UPPAAL in a Nutshell, Internatinal Journal on Software Tools for Technology Transfer, 1(1-2), 1997. - pp. 134–152.

[51] Lawvere F. W. The Category of categories as a foundation for mathematics// Proceedings of the Conference on Categorical Algebra. La Jolla: Springer-Verlag, 1996. - pp. 1–21.

[52] Lightfoot D. Formal Specification Using Z. 2nd Ed. Palgrave, 2001.

[53] Lopes D., Hammoudi S., Bezivin J. and Jouault F. Mapping Specification in MDA: From Theory to Practice, 2004// First International Conference INTEROP-ESA'2005 Interoperability of Enterprise Software and Applications. - Geneva, Switzerland: Springer, 2005. // Internet: http://Interop-esa05.unige.ch/INTEROP/Proceedings/Interop-ESAScientific/PerPaper/I08-2\%20354.pdf

[54] Mac Lane S. Categories for the Working Mathematician. Springer-Verlag, 1971.

[55] Mac Lane S. Dualities for Groups// Bulletin of the American Mathematical Society. Vol. 56, 1950. - pp. 485–516.

[56] McDonald J., Anton J. Specware — Producing Software Correct by Construction, March 2001. // Internet: ftp://ftp.kestrel.edu/pub/papers/specware/specware-jm.pdf

[57] Meseguer J., Montanari U. Petri Nets Are Monoids: A New Algebraic Foundation for Net Theory. 155-164// Proceedings, Third Annual Symposium on Logic in Computer Science, 5-8 July 1988, Edinburgh, Scotland, UK

[58] Milner R. Communication and Concurrency, Prentice-Hall, 1995.

[59] Murata T. Petri Nets: Properties, Analysis and Applications// In: Proceedings of the IEEE, Vol. 77, No. 4, April 1989. - pp. 541–580.

[60] Naur P., Randell B. (Eds.) Software Engineering: Report of a conference sponsored by the NATO Science Committee, Garmisch, Germany, 7-11 Oct. 1968, Brussels, Scientific Affairs Division, NATO, 1969. p. 231.// Internet: http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

[61] OMG Model Driven Architecture// Internet: http://www.omg.org/mda

[62] OMG Unified Modeling Language// Internet: http://www.uml.org/

[63] Pierce C. B. Basic Category Theory for Computer Scientists, 5th Ed., London: MIT Press, 2002. - p. 101.

[64] Pilone D., Pitman N. UML 2.0 in a Nutshell. A Desktop Qick Reference, O'Reilly Media, Inc., USA, California, 2005.

[65] Pnueli A. The Temporal Logic of Programs// In Proceedings of the 18th IEEE Symposium Foundations of Computer Science (FOCS 1977), 1977. - pp. 46–57.

[66] Process Specification Language.// Internet: http://www.mel.nist.gov/psl/index.html

[67] Rydeheard D.E., Burstall R.M. Computational Category Theory. Prentice Hall, 1988.

[68] Sekerinski E., Zurob R. Translating statecharts to B// In M. J. Butler, L. Petre, and K. Sere, editors, 3rd International Conference on Integrated Formal Methods (IFM), Lecture Notes in Computer Science, Springer-Verlag, May 2002. - pp. 128-144.

[69] Sheena R. Judson S., France R. and Carver D. Specifying Model Transformations at the Metamodel Level, 2003// Internet: http://www.metamodel.com/wisme-2003/19.pdf

[70] Specware 4.0 User Manual// Internet: http://www.specware.org/documentation/4.0.5/user-manual/SpecwareUserManual.pdf

[71] Smith G. The Object-Z Specification Language. Kluwer Academic Publishers, 2000.

[72] Smith J., DeLoach S., et. al. Category Theoretic Approaches of Representing Precise UML semantics// Internet: http://www.cis.ksu.edu/~sdeloach/publications/Conference/ecoop-pUML2000.pdf

[73] Smith J., Kokar M., Baclawski K. Formal Verification of UML Diagrams: A First Step Towards Code Generation// pUML 2001, Toronto, Ontario, Canada, 2001. - pp. 224–240.

[74] Specware Home Page// Internet: http://www.specware.org

[75] Spivey J. M. The Z Notation: A Reference Manual, Second Edition// Internet: http://spivey.oriel.ox.ac.uk/mike/zrm/index.html

[76] Wermelinger M., Fiadeiro L. J. Connectors for Mobile Programs// IEEE Transactions on Software Engineering, Vol. 24, No. 5, May 1998. - pp. 331–341.

[77] Wiels V., Easterbrook S. Management of evolving specifications using category theory// Internet: http://www.cert.fr/francais/deri/wiels/./Publi/ase98.ps

[78] Zhao Y., et al. Towards Formal Verification of UML Diagrams Based on Graph Transformation// In: Proceedings of the E-Commerce Technology For Dynamic E-Business International Conference,Volume 00, IEEE Computer Society, Washington, DC, USA, 2004. – pp. 180–187.// Internet: http://dx.doi.org/10.1109/CEC-EAST.2004.70

[79] Осис Я. Я. Топологическая модель функционирования систем// В. Журн. «Автоматика и вычислительная техника», Вып. 6, Рига: Зинатне, 1969. - 44.-50. сс.

[80] Осис Я. Я. Исследование нарушений функционирования сложных систем и теория категорий// Сборник трудов: Кибернетика и диагностика. Рига: Риж. политехн. ин-т, Вып. 4, 1970. - 15.-19. сс.

[81] Соммервилл И. Инженерия программного обеспечения, 6-е издание. Москва: Издательский дом «Вильямс», 2002. - 187.-202. сс.