KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

Donatas Savickis

Paskirstytųjų objektinių duomenų bazių transakcijų valdymo protokolo tyrimas

Magistro darbas

Darbo vadovas
prof. habil. dr. H. Pranevičius

Kaunas, 2005

KAUNO TECHNOLOGIJOS UNIVERSITETAS
INFORMATIKOS FAKULTETAS
VERSLO INFORMATIKOS KATEDRA

TVIRTINU
Katedros vedėjas
prof. habil. dr. H. Pranevičius

Paskirstytųjų objektinių duomenų bazių transakcijų valdymo protokolo tyrimas

Informatikos mokslo magistro baigiamasis darbas

Kalbos konsultantė
Lietuvių kalbos katedros lektorė
dr. J. Mikelionienė

Vadovas
prof. habil. dr. H. Pranvičius

Recenzentas
doc. dr. E. Bareiša

Atliko
IFM 9/1 gr. stud.
D. Savickis

Kaunas, 2005

SANTRAUKA

GLOBDATA yra Europos IST projektas, kurio tikslas – suprojektuoti ir realizuoti tarpinio lygmens įrankį, supaprastinantį paskirstytųjų objektinių duomenų bazių valdymą. Šis įrankis, vadinamas COPLA, leidžia manipuliuoti objektiniais duomenimis geografiškai nutolusiose sistemose, nepriklausomai nuo jų vietos. Be to, šis įrankis leidžia pasirinkti replikuojamų duomenų vientisumo modelį. Tam sukurti lengvai pakeičiami vientisumo protokolai, kurie leidžia pasirinkti reikiamą našumo ir pakantumo klaidoms santykį.

Šis darbas aprašo vieno iš dviejų vientisumo protokolų GLOBDATA sistemai, vadinamojo balsavimo protokolo, tyrimą. Protokolas sukurtas panaudojus atominį transliavimo protokolą kaip pagrindinę komunikavimo priemonę. Naudojantis UML priemonėmis, darbe išsamiai aprašoma konceptualiojo modelio analizė, o analizės rezultatų pagrindu aprašomas formalusis sistemos modelis. Formalusis modelis aprašomas naudojant paskirstytųjų sistemų formalizavimo ir analizės metodą, kurio teorinis pagrindas – atkarpomis tiesinių agregatų formalizavimas. Šis formalusis modelis ir yra darbo rezultatas, kuris gali būti toliau naudojamas protokolo tyrimui: protokolo validavimui, verfikavimui ir simuliacijai, įvairių protokolo charakteristikų tyrimui matematinių metodų pagrindu.

SUMMARY

GLOBDATA is a project that aims to design and implement a middleware tool offering the abstraction of a global object database repository. This tool, called COPLA, supports transactional access to geographically distributed persistent objects independent of their location. Additionally, it supports replication of data according to different consistency criteria. For this purpose, COPLA implements a number of consistency protocols offering different tradeoffs between performance and fault-tolerance.

This paper presents the analysis of one of two strong consistency protocols for the GLOBDATA system, the voting protocol. A protocol relies heavily on the use of atomic translation primitive as a building block to serialize conflicting transactions. The paper presents an in-depth description of the conceptual model using UML-based analysis of the protocol which later is used to define the PLA (piece-linear aggregates) model of the system, a discrete event-based simulation of the processes. As a result of the work, the PLA model is ready to be used to validate, verify and simulate various behavioral and performance characteristics of the model using mathematical proof techniques.

TURINYS

PAVEIKSLAI

1 ĮVADAS

Duomenų replikavimas paskirstytosiose duomenų bazėse dažniausiai naudojamas kaip priemonė siekiant padidinti duomenų prieinamumą ir taikomųjų programų našumą. GLOBDATA yra Europos IST projektas, kurio tikslas - sukurti ir realizuoti tarpinio lygmens įrankį, padėsiantį supaprastinti paskirtytųjų objektinių duomenų bazių valdymą. Šis įrankis, vadinamas COPLA, leidžia logiškai atskirtuose ir geografiškai nutolusiuose mazguose naudoti transakcijas operacijoms su paskirstytaisiais objektais nepriklausomai nuo geografinio mazgų išsidėstymo. Taikomųjų programų kūrėjai gali manipuliuoti objektiniais duomenimis ir jiems nereikia rūpintis tuo, kaip objektai saugomi, paskirstomi arba replikuojami. Tuo pačiu COPLA tarpinis lygmuo palaiko duomenų replikavimą tenkindamas skirtingus vientisumo modelius. Kiekvienas modelis gali būti realizuojamas vienu ar keliais vientisumo protokolais skirtingai konfigūruojant sistemą ir renkantis kompromisus tarp greičio ir klaidų tikimybės (pakantumo klaidoms).

Prieš kuriant sistemas, būtina jas suprojektuoti ir ištirti, tam yra naudojamos įvairios priemonės ir metodikos. Tai ypač svarbu, kai sistemos yra sudėtingos, arba jos vykdo kritines funkcijas. Tais atvejais, kuomet projektą vykdo komanda arba bendradarbiauja kelios skirtingos komandos, būtina vienareikšmiškai aprašyti sistemos funkcionavimą bei prielaidas kuriomis remiantis turi būti kuriama sistema. Tai vienas iš sistemų projektavimo ir modeliavimo tikslų.

Be to, būtina užtikrinti, kad sistema veikia teisingai, kad jos veikimo metu nesusidarys situacijos, kuomet sistema pradės veikti ne taip, kaip buvo numatyta. Be abejo visuomet vykdomas galutinės sistemos testavimas, tačiau paprastai neįmanoma numatyti ir išmėginti visų galimų sistemos algoritmo vykdymo kelių. Ypač sudėtinga tai padaryti sistemose, kurių komponentai vykdomi lygiagrečiai. Tokių sistemų projektavimas, kūrimas ir veikimas yra sudėtingesnis dėl įvairių galimų situacijų, kurios gali susidaryti skirtingiems procesams naudojant tuos pačius resursus arba kuomet procesams reikia sinchronizuoti savo veiksmus.

Taigi, norint užtikrinti patikimą sistemų veikimą ir integraciją, taip pat norint minimizuoti darbo sąnaudas sistemų kūrimui, būtina ištirti sistemos komponentų savybes dar prieš realizuojant sistemą. Šiame darbe aptariamai sistemai kurti naudojamos teorinės prielaidos ir priemonės yra naujos, daugelis jų yra menkai ištirtos arba ištirtos tik empirinėmis priemonėmis. Todėl ypač svarbu išmėginti sistemą ir išnagrinėti jos

charakteristikas naudojant matematinius modelius. Sistemų tyrimas teoriniame lygmenyje naudojant matematines schemas ir priemones leidžia objektyviai bei universaliai patikrinti jų charakteristikas. Tačiau norint ištirti sistemą, pirmiausia būtina sukurti jos modelį.

Darbas aprašo vienos GLOBDATA projekto dalies - vientisumo protokolo - tyrimą. Čia tiriamas vienas iš dviejų stipraus vientisumo protokolų, vadinamasis balsavimo protokolas. Protokolas remiasi kitomis GLOBDATA projekto priemonėmis, kurios aptariamos ir tiriamos tiek, kiek to reikia vientisumo protokolui tirti.

Pagrindinis darbo tikslas – surinkti, išanalizuoti ir apibendrinti medžiagą, aprašančią balsavimo protokolą bei jo veikimo kontekstą, ištirti konceptualųjį sistemos modelį ir sukurti formalųjį jo modelį. Sukurtą modelį vėliau būtų galima naudoti sistemos validavimui, verifikavimui bei simuliavimui. Taigi, darbo tikslas - aprašyti protokolą naudojantis paskirstytųjų sistemų formalizavimo ir analizės metodą, kurio teorinis pagrindas – atkarpomis tiesinių agregatų formalizavimas. Ši matematinė schema leidžia bendrosios formaliosios specifikacijos pagrindu validuoti sudarytą specifikaciją (atlikti teisingumo analizę) bei sudaryti analizuojamos sistemos imitacinį modelį. Norint supaprastinti formalizavimo procesą ir plačiau susipažinti su protokolo savybėmis, sistemos architektūrai aprašyti naudojama standartinė UML (angl. Unified Modeling Language*) grafinių notacijų kalba.*

Darbas susideda iš tokių dalių: probleminės srities aprašymas, konceptualiojo modelio aprašymas (susidedantis iš algoritmo aprašo bei UML diagramų), protokolo agregatinė specifikacija, darbo išvados ir trumpa galimų tolesnių tyrimų apžvalga.

2 GLOBDATA PROJEKTO ARCHITEKTŪRA

Didėjanti verslo globalizacija bei interneto ir WAN tinklų plitimas ir naudojimas verčia įmones ir įstaigas naudoti ir saugoti didelius duomenų kiekius paskirstytosiose duomenų bazėse, kurios išdėstomos skirtinguose vietose – miestuose, šalyse ir net žemynuose. Šiuos duomenis skirtingi vartotojai, kurie taip pat dažnai būna išsidėstę skirtingose geografinėse vietose, privalo apdoroti tuo pačiu metu. Toks dalijimasis bei manipuliavimas duomenimis nėra tradicinių duomenų bazių savybė. Iš kitos pusės, pastaruoju metu ypač išpopuliarėję objektinio projektavimo metodai leidžia efektyviau kurti duomenų apdorojimo sistemas, tačiau reikalauja naujų instrumentinių priemonių duomenų apdorojimui. Šiuolaikinės objektinės duomenų bazės kuriamos atsižvelgiant į šiuos naujus reikalavimus. Tačiau būtina atkreipti dėmesį ir į tai, kad didžiuliai duomenų kiekiai šiuo metu saugomi sąsajinėse duomenų bazėse, joms sukurta ir kuriama daugybė taikomųjų programų. Tiesiog pakeisti šių sistemų neįmanoma, todėl vienintelis būdas – skaidri integracija ir bendradarbiavimas.

Pagrindinis GLOBDATA projekto tikslas – sukurti ir realizuoti efektyvų tarpinio lygmens įrankį, pavadintą COPLA, suteikiantį lengvą priėjimą prie replikuojamo objektų repozitoriumo. Įrankis programuotojui turi pateikti objektinį duomenų vaizdą bei leisti vykdyti operacijas su šiais geografiškai nutolusiais objektiniais duomenimis nepriklausomai nuo jų vietos transakcijų kontekste. Duomenų replikos gali būti saugomos skirtinguose klasterio, vietinio tinklo arba net geografiškai nutolusiuose tinklo mazguose.

2.1 DUOMENŲ BAZIŲ TRANSAKCIJOS

Duomenų bazės transakcija yra bendravimo su duomenų bazės valdymo sistema (DBVS) vienetas, kuris vykdomas nuosekliai ir patikimai, nepriklausomai nuo kitų lygiagrečiai vykdomų transakcijų. Idealiu atveju DBVS turi garantuoti visas ACID (angl. *Atomicity, Consistency, Isolation, Durability*) savybes: atomiškumą, vientisumą, atskyrimą ir ilgalaikiškumą. Praktikoje šios savybės dažnai iš dalies aukojamos, kad būtų užtikrintas didesnis sistemų našumas.

Transakcijų naudojimas DBVS produktuose vartotojams garantuoja, kad bus palaikomas duomenų korektiškumas.

Vienos transakcijos metu gali būti reikalingos kelios užklausos, kurios skaito ir (arba)

keičia informaciją duomenų bazėje. Tokiais atvejais itin svarbu įsitikinti kad duomenų bazėje buvo įvykdytos visos užklausos, o ne tik kai kurios iš jų. Pavyzdžiui vykdant pinigų pervedimą svarbu užtikrinti, kad jei pinigai buvo nuskaičiuoti iš vienos sąskaitos, tai jie tikrai atsiras kitoje sąskaitoje. Taip pat transakcijos negali trukdyti viena kitai.

Paprastai transakcijos vykdomos tokia tvarka, naudojant užklausų kalbą:

1. Transakcija pradedama;
2. Įvykdomos kelios užklausos (tačiau pakeitimai, atlikti šioje transakcijoje, nematomi išorėje, t.y. transakcijos pakeitimai matomi tik šios transakcijos kontekste);
3. Transakcija užbaigiama (angl. *commit*) surašant duomenis į duomenų bazę (t.y. pakeitimai, atlikti šioje transakcijoje, atsiranda ir išorėje).

Jei transakcija nutrūko arba buvo specialiai nutraukta (angl. *abort*) bet kuriame žingsnyje prieš ją užbaigiant, DBVS atšauks (angl. *rollback*) visus pakeitimus ir duomenys liks nepakeisti. Kitos transakcijos veiks tarsi šios transakcijos nė nebūtų buvę.

Tam tikrais atvejais dvi lygiagrečiai vykdomos transakcijos gali persidengti - t.y. joms gali prireikti nuskaityti arba pakeisti tuos pačius duomenis. Tuomet sakoma kad transakcijos konfliktuoja. Iš principo dvi lygiagrečiai vykdomos transakcijos gali persidengti trim būdais: mėginant nuskaityti tą patį įrašą; kai viena mėgina nuskaityti, o kita - pakeisti tą patį įrašą; kai abi mėgina pakeisti tą patį įrašą. Problemos iškyla kai viena arba abi transakcijos mėgina pakeisti tą patį įrašą. Tuomet būtinas konfliktų sprendimo mechanizmas. Tradicinėms sąsajinėms DBVS sukurta įvairių konfliktų sprendimo būdų, tačiau jie ne visuomet tinka paskirstytosioms duomenų bazėms.

Naudojant paskirstytąsias duomenų bazes, konfliktų sprendimą komplikuoja tai, kad replikuojami duomenys apdorojami keliose skirtingose vietose, todėl būtina sinchronizuoti konfliktų sprendimą tarp skirtingų duomenų replikų. Paskirstytoji transakcija (angl. *distributed transaction*) yra veiksmų rinkinys, kuriame dalyvauja du arba daugiau į tinklą sujungtų sistemų.

2.2 PASKIRSTYTOSIOS DUOMENŲ BAZĖS

Paskirstytoji sistema – tai tokia sistema, kuri susideda iš komponentų, tuo pat metu veikiančių skirtinguose kompiuteriuose. Šie komponentai bendrauja tarpusavyje ir yra sukurti veikti atskirai vienas nuo kito. Pagrindinis paskirstytųjų sistemų tikslas yra suteikti vartotojams skaidrų (angl. *transparent*) priėjimą prie resursų, kuris būtų lengvai plečiamas (angl. *scalable*). Toks būdas taip pat yra žymiai galingesnis ir labiau pakantus klaidoms

(angl. *fault tolerant*).

Paskirstytosios duomenų bazės yra vienas iš paskirstytųjų sistemų pavyzdžių, kai duomenys yra paskirstomi tarp skirtingų kompiuterių. Paprastai duomenų bazės yra replikuojamos, t.y. sukuriamos identiškos jų kopijos. Taip gali būti padidintas duomenų patikimumas (kadangi egzistuoja daugiau nei viena duomenų kopija), sistemos našumas (kadangi duomenys prieinami daugelyje vietų). Žinoma, tokiu atveju iškyla daug papildomų problemų - duomenų sinchronizacija, vientisumo užtikrinimas, sistemos komponentų sutrikimų apdorojimas, komunikacijos problemos.

2.3 VIENTISUMO MODELIAI

Paskirstytosiose sistemose, taip pat ir paskirstytosiose duomenų bazėse, būtina užtikrinti duomenų vientisumą (angl. *data consistency*). Duomenų vientisumas reiškia kad prasidedant ir pasibaigus transakcijai sistemos duomenų būsena yra korektiška. Tai reiškia, kad pasibaigus transakcijai sistemos duomenys bus teisingi ir atitiks taisykles, apibrėžtas sistemoje - nebus dalinai pakeistų, neteisingų, neatitinkančių taisyklių duomenų.

Užtikrinant duomenų teisingumą naudojami skirtingi vientisumo modeliai. Sakoma, kad sistema palaiko tam tikrą vientisumo modelį, jei operacijos su duomenimis laikosi tam tikrų taisyklių, kitaip tariant, užtikrinama kad, jei programuotojas laikysis tam tikrų taisyklių, kurias apibrėžia sistema, tuomet jis gali tikėtis, kad duomenys bus vientisi ir operacijų rezultatai bus nuspėjami.

Griežtas vientisumo modelis (angl. *strict consistency*) nurodo, kad tam tikro duomenų įrašo skaitymo operacijos visuomet turi grąžinti paskutinės rašymo operacijos rezultatą. Tačiau pagal fizikos dėsnius toks modelis negali būti naudojamas paskirstytosiose sistemose (informacija turėtų keliauti šviesos greičiu), nors puikiai tinka paprastose, vieno procesoriaus, sistemose. Todėl paskirstytosiose sistemose naudojami paprastesni, ne tokie griežti modeliai.

Stipraus vientisumo modelis nusako, kad duomenys turi būti įrašomi skirtinguose mazguose sinchroniškai ir nuosekliai (ta pačia tvarka) arba pastoviai atnaujinami (sinchronizuojami) skirtinguose mazguose. Tuomet duomenų teisingumas bus užtikrintas logine prasme – vykdant tas pačias operacijas skirtinguose mazguose, bus gaunamas toks pat rezultatas. Tačiau tai nereiškia, kad bus pasiekta absoliuti sinchronizacija laike.

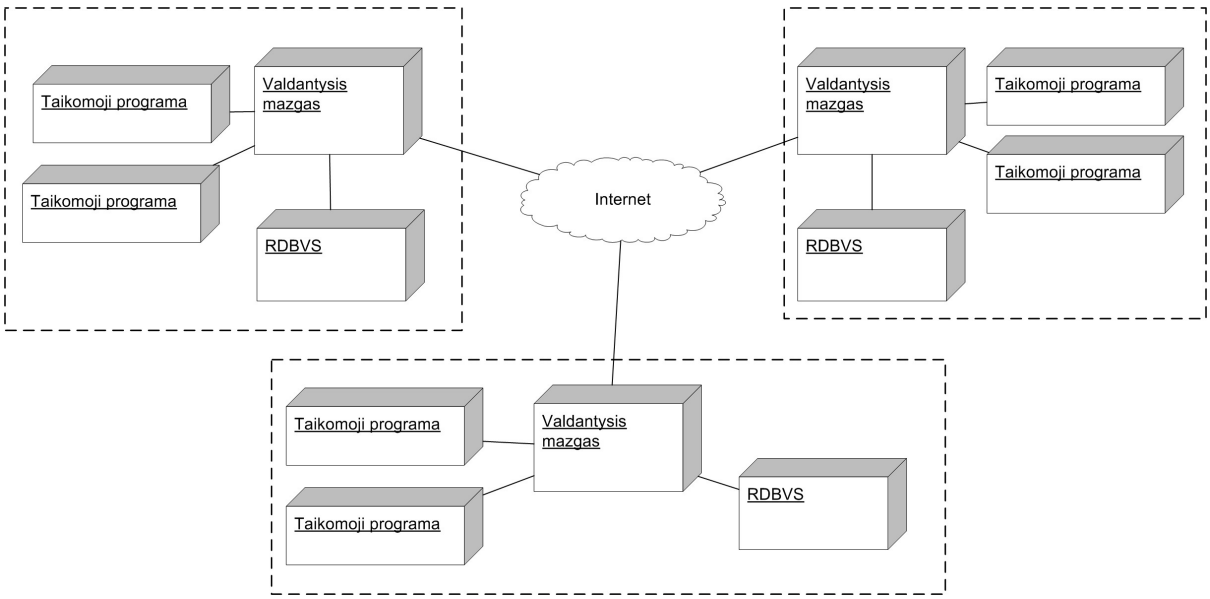
2.4 ARCHITEKTŪRINIAI GLOBDATA PROJEKTO TIKSLAI

Pagrindiniai GLOBDATA projektui keliami reikalavimai:

- Didelis mastelis. Vientisumo protokolai turi palaikyti objektų replikavimą geografiškai išsklaidytose sistemose, kuriose mazgai bendrauja įvairiais internetui naudojamais protokolais. Toks reikalavimas neleidžia išskirtinai taikyti specialios paskirties tinklo protokolų, kurie pasižymi specifinėmis savybėmis (pvz., mažo latentiškumo, siuntimo tvarką garantuojančių tinklo protokolų) .
- Nepriklausomybė nuo DBVS. Turi būti palaikomos įvairios standartinės komercinės DBVS, vengiant DBVS architektūros pritaikymo šiam uždaviniui.
- Protokolų pakeičiamumas. COPLA turi būti suprojektuota lanksčiai kad galėtų prisitaikyti (arba būti pritaikyta) besikeičiančioms aplinkos sąlygoms (sistemos mastelis, apkrova ir pan.). Taigi sistema turi palaikyti skirtingus vientisumo protokolus, turinčius skirtingas savybes skirtingose situacijose.
- Objektinis orientavimas. Nepriklausomai nuo to, kad COPLA objektus saugo sąsajinėje duomenų bazėje, ši operacija yra atskirta nuo vientisumo protokolų. Tokiu būdu algoritmai nepririšami prie konkretaus objekto atvaizdavimo.

2.5 VEIKIMO SCENARIJUS

COPLA sistemos architektūrinis modelis sukurtas pagal scenarijus, numatančius sistemos panaudojimo būdus, jos išdėstymo ir konfigūravimo poreikius.



1 pav. COPLA mazgų išdėstymo schema

Tinklas kuriame veikia sistema, skirstomas į zonas, susidedančias iš mazgų grupių. Mazgai, esantys fiziškai arti vienas kito, priklauso tai pačiai zonai. Pavyzdžiui tai galėtų būti

didelė įmonė, turinti filialus skirtinguose miestuose, o kompiuteriai skirtinguose filialuose priklauso tai pačiai zonai. Projektuojant sistemą, atitinkančią šį modelį, buvo sukurta tokia struktūra:

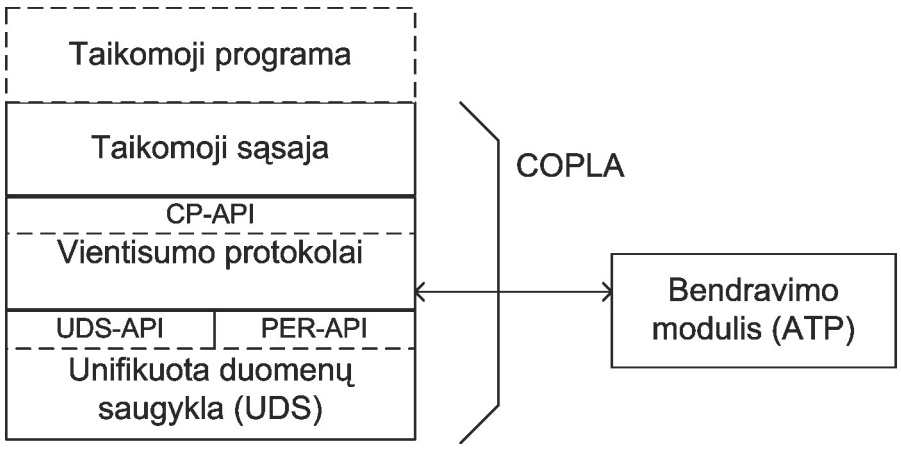
- Sistemoje veikia specialūs mazgai, vadinami valdančiaisiais mazgais. Kiekvienoje zonoje veikia vienas toks mazgas. Valdantysis mazgas atsako už COPLA taikomųjų programų aptarnavimą savo zonoje.
- Taikomosios programos bendrauja tik su artimiausiu valdančiuoju mazgu, t.y. savo zonos valdančiuoju mazgu. Kiekvienas valdantysis mazgas naudoja nuosavą duomenų bazę kaip priemonę duomenims saugoti. Daugelio duomenų bazių egzistavimo faktas (po vieną duomenų bazę kiekvienai sričiai) neturi jokios įtakos taikomosios programos vykdymui. Taikomoji programos darbas organizuotas taip, tarsi egzistuotų vienintelė duomenų bazė.
- Teoriškai visos duomenų bazės tinkle turi identišką visų objektų duomenų kopiją (repliką). O praktiškai vientisumo protokolo algoritmas nusprendžia, kokia informacija saugoma kiekvienoje duomenų bazėje.
- Valdantieji mazgai bendrauja vientisumo protokolais ir taip koordinuoja savo veiksmus. Tai leidžia:
    - teikti teisingus objektų duomenis taikomosioms programoms;
    - dalintis transakcijų, kurias vykdo taikomosios programos, rezultatais;
    - spręsti lygiagrečių transakcijų, vykdomų toje pačioje arba skirtingose srityse, konfliktus;
    - užtikrinti tam tikrą pakantumą klaidoms naudojantis tuo, kad duomenys replikuojami.
- Transakcijas, inicijuotas taikomosios programos, vykdo atitinkamos srities valdantysis mazgas, naudodamas tik to mazgo duomenų bazėje esančius duomenis (vėlgi naudojamas vientisumo protokolas nulemia visas smulkmenas).
- Jei valdančiojo mazgo darbas sutrinka, sistema turi toliau funkcionuoti, kiek tai įmanoma, tačiau privalo užtikrinti duomenų teisingumą ir vientisumą:
    - taikomosios programos, kurias aptarnavo sutrikęs valdantysis mazgas, negalės užbaigti transakcijų iki tol, kol mazgo darbas bus atstatytas;

- jei sutrikęs valdantysis mazgas buvo vienintelis mazgas, kuriame buvo saugoma paskutinė objekto versija, taikomosios programos, kurioms reikia to objekto duomenų, bus sustabdytos iki tol, kol mazgo darbas bus atstatytas. Vientisumo protokolai darys viską, kad būtų daugiau nei viena naujausia kiekvieno objekto duomenų kopija.

2.6 COPLA SISTEMOS KOMPONENTAI

Pagrindiniai COPLA valdančiojo mazgo komponentai pavaizduoti paveiksle. Viršutinis lygis yra taikymo sąsajos modulis, kurį pasiekia ir naudoja taikomosios programos. Tokių programų kūrėjas naudoja objektinę saugomų duomenų interpretaciją: duomenų manipuliavimui naudojama OQL (Object Query Language) kalbos modifikacija. Tie patys objektai pasiekiami lygiagrečiai tik paskirstytųjų transakcijų kontekste.

Objektinė duomenų bazė gali būti replikuojama tam, kad pagerinti pakantumą klaidoms, o taip pat tam, kad būtų galima skaityti duomenis iš lokaliosios saugyklos taupant tinklo resursus. Tai reiškia, kad identiškos duomenų kopijos saugomos kiekviename nutolusiame mazge. Tačiau duomenų replikavimas sukuria ir nemažai problemų – sudėtingesnį valdymą, sumažėjusį rašymo pralaidumą, poreikį spręsti konfliktines situacijas kai tie patys duomenys keičiami skirtingose vietose. Duomenų replikavimui COPLA palaiko keletą protokolų, vadinamų vientisumo protokolais (angl. *consistency protocols*). Jie gali būti pasirenkami priklausomai nuo tinklo topologijos, sistemos apkrovimo, reikalingų protokolo charakteristikų. Visi šie protokolai yra abstrahuojami bendra protokolų sąsaja CP-API ir taip išlaikomas vartotojo kodo atskyrimas nuo pasirinkto protokolo. Toks atskyrimas leidžia lanksčiai konfigūruoti COPLA įrankį ir keisti vientisumo protokolus, priklausomai nuo veikimo aplinkos charakteristikų, bet nekeičiant kitų sistemos dalių.



2 pav. COPLA įrankio architektūra

Kaip ir parodyta 2 paveiksle, sistema sudaryta iš tokių pagrindinių modulių:

- Taikymo sąsaja (angl. *Client Interface*) – modulis, kurį mato COPLA įrankį naudojantys taikomųjų programų kūrėjai. Šis modulis atsakingas už sąsajos pateikimą taikomųjų programų kūrimui ir už komunikavimą su kitais dviem moduliais (vientisumo protokolais ir UDS (unifikuota duomenų saugykla)). Taikomosios programos tiesiogiai bendrauja su šiuo moduliu.
- Vientisumo protokolai (angl. *Consistency Protocols*) – modulis, atsakingas už replikuojamų duomenų teisingumą ir vientisumą skirtingose replikose. Vientisumo protokolai bendrauja su kitais valdančiaisiais mazgai bendravimo modulio pagalba (paprastai ATP). Skirtingi protokolai pateikia vienodą sąsają (pavadintą CP-API). Tai leidžia konfigūruoti COPLA įrankį priklausomai atsižvelgiant į reikalingas charakteristikas ir aplinką kurioje sistema veikia.
- Unifikuota Duomenų Saugykla (angl. *Uniform Data Store, UDS*) – modulis, atsakingas už objektų būsenų saugojimą. Ši duomenų saugykla pateikia vartotojui objektinį duomenų vaizdą nepriklausomai nuo to kur ir kaip saugomi duomenys standartinėse DBVS. UDS pateikia UDS-API sąsają, naudojamą nuskaityti ir saugoti objektus, versti taikomosios programos užklausas į SQL užklausas neprisirišant prie naudojamos DBVS. UDS taip pat pateikia PER-API sąsają, kurią naudoja vientisumo protokolai nuskaityti ir saugoti protokolų valdymo informaciją.
- Bendravimo modulis (angl. *Communication Module*) – modulis, atsakingas už valdančių mazgų bendravimą. Abu stipraus vientisumo protokolai, skirti COPLA įrankiui, buvo sukurti remiantis atominio transliavimo protokolo (ATP) savybėmis. ATP protokolas yra absoliučios tvarkos (angl. *total order*) grupinio bendravimo (angl. *group communication*) šeimos protokolas. Pati protokolo realizacija yra išties sudėtinga ir jo nagrinėjimas galėtų būti atskiro darbo tema. Šiame darbe svarbios tik protokolo teikiamos paslaugos ir jo charakteristikos, o ne veikimo principas.

2.7 STIPRAUS VIENTISUMO PROTOKOLAI

Kaip ir minėta, COPLA sistemos vartotojai turi galimybę rinktis tarp protokolo tolerancijos klaidoms ir našumo. Todėl kuriami protokolai tenkinantys skirtingus poreikius. Lisabonos Universitetas pasiūlė du stipraus vientisumo (angl. *strong consistency*) protokolus – nebalsavimo ir balsavimo. Abu šie protokolai remiasi atominio transliavimo protokolu

(angl. *atomic broadcast primitive, ATP*), kuris čia taip pat bus aptartas. Šio darbo tikslas – sukurti agregatinį modelį vienam iš protokolų ir jį ištirti. Tai vadinamasis balsavimo (angl. *voting*) protokolas.

2.7.1 SĄVEIKA TARP KOMPONENTŲ

Vientisumo protokolai keičiasi informacija su UDS apie tai, kurie objektai yra nuskaitomi, o kurie įrašomi kiekvienos transakcijos metu. Ši informacija, apdorojama kaip objektų unikalių identifikatorių (OID) sąrašas, leidžia protokolams nustatyti kurios transakcijos konfliktuoja tarpusavyje. Kadangi patys vientisumo protokolai apdoroja tik identifikatorius, jie išlieka nepriklausomi nuo objektų atvaizdavimo duomenų bazėje.

2.7.2 COPLA TRANSAKCINIS MODELIS

COPLA sistemoje transakcijos vykdymas susideda iš tokių žingsnių:

1. Programuotojas informuoja sistemą apie transakcijos pradžią.
2. Programuotojas vykdo užklausą į duomenų saugyklą kuri gražina objektų sąrašą.
3. Gražinti objektai apdorojami naudojant klientinės sąsajos pateiktas funkcijas. Šios funkcijos leidžia keisti objektų savybes bei nuskaityti papildomus objektus, tam tikrais ryšiais susijusius su apdorojamu objektu (pvz. kai objektas turi o nuorodą į kitą objektą).
4. 2 ir 3 žingsniai kartojami tol, kol transakcijos veiksmai užbaigiami.
5. Programuotojas informuoja sistemą apie transakcijos pabaigą.

2.7.3 SĄVEIKA SU VIENTISUMO PROTOKOLAIS

Bendra protokolų sąsaja CP-API pateikia dvi funkcijas, kurios leidžia apsikeitimą informacija su vientisumo protokolu:

- *UDSAccess(t, l)*, kur t – transakcija, l – objektų sąrašas. Ši funkcija skirta:
  a) užtikrinti, kad lokaliosios objektų kopijos yra naujausios;
  b) gauti objektų būsenas, nuskaitant jas iš UDS (duomenų skaitymas realizuojamas ne paties protokolo, o naudojama UDS funkcija);
- *commit()* funkcija naudojama taikomosios programos transakcijai užbaigti.

Reikia paminėti, kad tikroji funkcijos *UDSAccess()* realizacija gali būti išskaidytą į eilę smulkesniu funkcijų, vykdančių panašias operacijas (atributų nuskaitymas, ryšių nuskaitymas, užklausos ir t.t.).

Iškvietus commit() funkciją, vientisumo protokolas su kitais mazgais turi suderinti savo sprendimą apie tai, ar transakciją galimą saugiai užbaigti ir įrašyti duomenis, ar transakciją reikia nutraukti dėl konflikto. Šios fazės vykdymo metu protokolas iš UDS užklausia objektų, kurie buvo pakeisti transakcijos metu, sąrašo. Taip pat UDS pateikia protokolui objektų būsenų pasikeitimus. Protokolas taip pat turi užtikrinti šių pasikeitimų persiuntimą kitiems mazgams.

2.7.4 ATOMINIO TRANSLIAVIMO PROTOKOLAS

COPLA sukurti stipraus vientisumo protokolai remiasi vienu iš grupinio bendravimo protokolų, aprašytu [1]. Atominio transliavimo protokolas (angl. *Atomic Translation Privitive, ATP*) perduota pranešimus tarp serverių grupės narių ir užtikrina pristatymo atomiškumą bei tvarką.

Protokolo veikimo principas parodytas 3 paveiksle. Pristatymo atomiškumo charakteristika garantuoja, kad jei grupės *g* nariui buvo pristatytas pranešimas *m*, tuomet visiems grupės nariams buvo pristatytas pranešimas *m*. Pristatymo tvarkos charakteristika užtikrina, kad jei bet kuriems dviem grupės *g* nariams buvo pristatyti pranešimai *m* ir *m′*, tuomet jie pristatyti ta pačia tvarka. Šios dvi savybės yra naudojamos vientisumo protokolų: tvarkos užtikrinimas naudojamas konfliktų sprendimui, o atomiškumas leidžia supaprastinti transakcijų valdymą.

Pats protokolo algoritmas nėra šio darbo tyrimo objektas. Vientisumo protokolai buvo sukurti naudojant atominį transliavimo protokolą (tiksliau dvi jo savybes) kaip bendravimo būdą tarp grupės narių.

3 BALSAVIMO PROTOKOLAS

Balsavimo protokolas, aprašytas [2] yra protokolo, aprašyto [3], pritaikymas COPLA transakciniam modeliui. Bendras algoritmas yra toks:

1. Visos transakcijų operacijos yra vykdomos lokaliai, vykdančiajame mazge. Nuskaitomiems objektams gaunami lokalieji skaitymo užraktai, o fizinės rašymo operacijos atidedamos iki tol, kol transakcija iki galo baigs darbą. Būtina pažymėti kad objektas gali būti įrašytas tik tuo atveju, jei jis prieš tai buvo nuskaitytas.
2. Kai taikomoji programa nusprendžia pakeitimus išsaugoti (angl. c*ommit*), pakeitimų sąrašas yra išsiunčiamas visiems grupės mazgams (įskaitant ir vykdantįjį mazgą) atominiu transliavimo primityvu.
3. Kai transakcijos *t* pakeitimų sąrašas pristatomas, visi dalyvaujantys mazgai siekia gauti lokaliuosius rašymo užraktus visiems objektams sąraše. Jei egzistuoja transakcija, kuri turi rašymo užraktą vienam iš objektų, tuomet *t* vykdymas atidedamas iki tol, kol rašymo užraktas bus atlaisvintas. Transakcijos, turinčios skaitymo užraktus yra nutraukiamos, išsiunčiant transakcijos nutraukimo žinutę ATP. Kai vykdantysis mazgas gauna visus rašymo užraktus, jis išsiunčia transakcijos užbaigimo žinutę ATP.
4. Gavus transakcijos užbaigimo žinutę, mazgai surašo pakeitimus į lokaliąsias duomenų bazes ir atlaisvina visus užraktus. Gavus transakcijos nutraukimo žinutę, visi užraktai atlaisvinami, o vykdantysis mazgas nutraukia transakciją.

3.1 REPLIKAVIMO STRATEGIJA

Balsavimo protokolas gali būti priskirtas prie „atnaujinančiųjų visur pastoviąja sąveiką“ (angl. *update everywhere constant interaction*) replikavimo strategijų. Protokolas yra „atnaujinantysis visur“ tipo, nes duomenys atnaujinami visose sistemos replikose vienu metu. Tokia savybė pasirinkta norint siekiant apdoroti klaidas (visi mazgai turi savo duomenų kopijas), be to šitaip išvengiama komunikacijos kanalų perkrovimo vienoje vietoje (vadinamasis „butelio kakliuko“ efektas). Jis yra „pastoviosios sąveikos“ protokolas, kadangi transakcijos pranešimų skaičius yra baigtinis ir fiksuotas (žinomas iš anksto) nepriklausomai nuo transakcijoje vykdomų veiksmų skaičius.

3.2 REPLIKAVIMAS NAUDOJANT ATOMINĮ TRANSLIAVIMĄ

Panaudojus grupinio bendravimo, t.y. atominio ir virtualiai sinchroniško transliavimo, protokolą kaip pagrindą kuriant duomenų replikavimo sistemą, gerokai supaprastėja duomenų vientisumo, pakantumo klaidoms ir užraktų valdymo mechanizmų sudėtingumas.

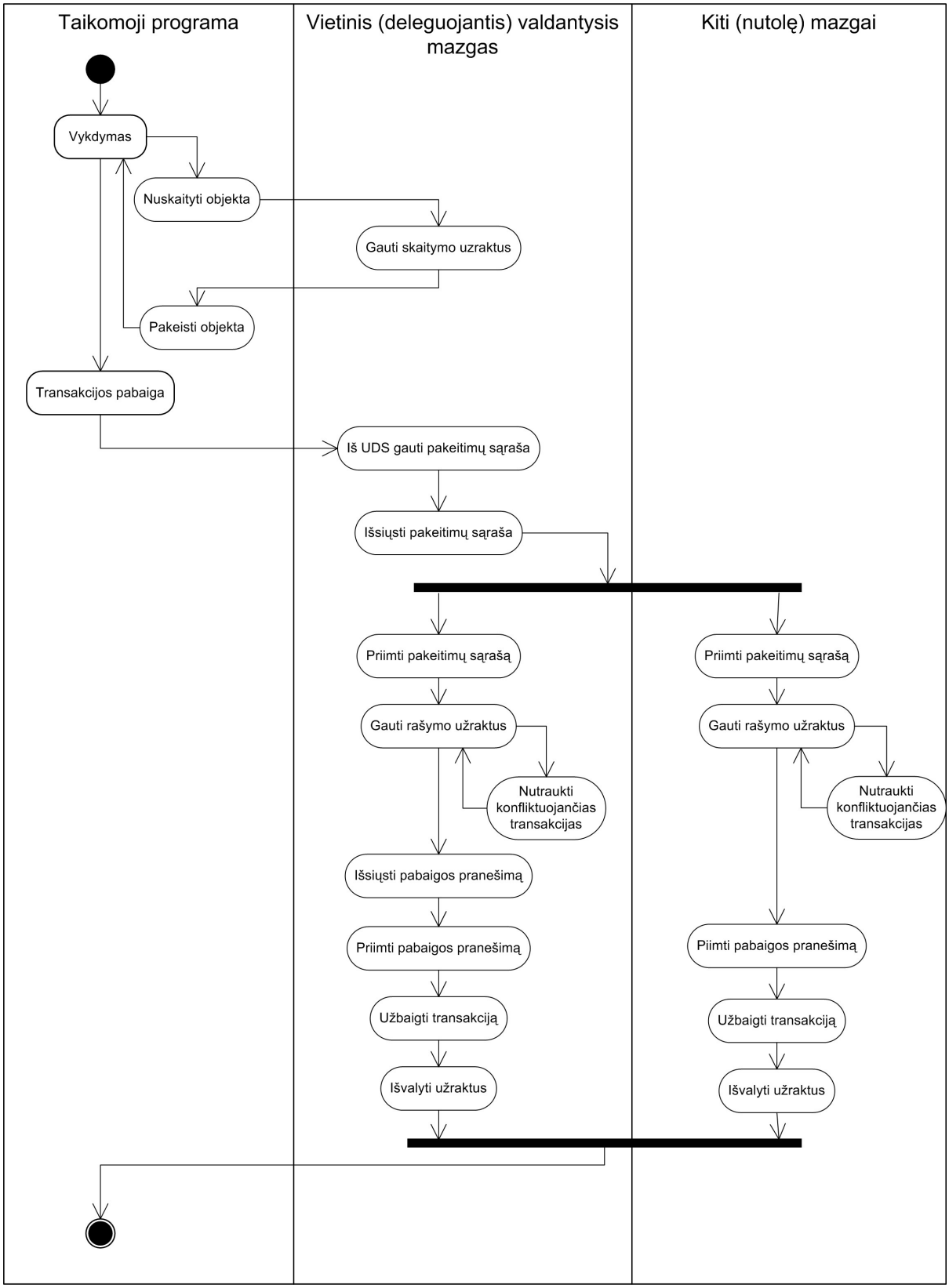
Transakcijomis paremta sistema taip pat turi įgyvendinti ACID savybes: atomiškumą (angl. *atomicity*), vientisumą (angl. *consistency*), atskyrimą (angl. *isolation*) ir ilgalaikiškumą (angl. *durability*). Duomenų bazių replikavimas sukuria papildomų problemų įgyvendinant šias savybes. Toliau bus apžvelgta kaip algoritmas įgyvendina šias savybes.

Kad tenkintų atomiškumo savybę, sistema turi užtikrinti šią savybę visuose mazguose – įvykdomos arba visos transakcijos operacijos, arba atsiradus sutrikimams neįvykdoma nė viena. Šis rezultatas turi būti vienodas visuose mazguose. Kiekvienai transakcijai COPLA vykdo operacijų seką lokaliosios transakcijos kontekste deleguojančiame mazge (t.y. valdančiame mazge, jungiančiame taikomąją programą su kitais sistemos mazgais). Kai programa bando užbaigti transakciją, operacijų seka išsiunčiama visiems mazgams viena žinute, kuri negali būti pristatyta tik dalinai. Užtikrinimą transakcijos atomiškumą deleguojantis mazgas įpareigoja kiekvieną sistemai priklausantį valdantį mazgą.

Vientisumo savybė įpareigoja taip vykdyti persidengiančias transakcijas, kad jų vykdymas būtų lygus nuosekliam transakcijų vykdymui tam tikra tvarka. Paskirstytosiose duomenų bazėse būtina užtikrinti ne tik lokaliai vykdomų transakcijų vientisumą, bet ir transakcijų, lygiagrečiai vykdomų skirtinguose mazguose, vientisumą. Balsavimo protokolas surašo transakcijų pakeitimus naudodamasis atominiu transliavimo protokolu, kuris užtikriną tikslią pranešimų pristatymo tvarką skirtingiems sistemos mazgams. Algoritmas sukurtas tikintis tikslios tvarkos pristatant pranešimus net tuo atveju, kai įvyksta klaida kuriamo nors mazge.

Lygiagrečių transakcijų atskyrimo savybe užtikrinama vietiniame mazge, modifikuojant duomenis tik transakcijos kontekste ir atidedant pakeitimų surašymą iki kol transakcijos veiksmai bus užbaigti. Visos kitos taikomosios programos, įskaitant ir tas, kurios bendrauja su nutolusiais valdančiaisiais mazgais, nežino apie šių neužbaigtų transakcijų egzistavimą.

Ilgalaikiškumas užtikrinamas DBVS lokaliai kiekviename mazge. Kai išėjęs iš rikiuotės mazgas atstatomas, jis pirmiausia bus sinchronizuotas su kitais mazgais, užtikrinant kad jam bus pranešta apie kiekvieną baigtą transakciją.



3 pav. Transakcijos veiklos diagrama

4 BALSAVIMO PROTOKOLO MODELIS

Paskirstytosios sistemos gali būti tiriamos dviem požiūriais: elgsenos (angl. *behaviour*) ir funkcionavimo (angl. *performance*). Elgsenos analizės metu tiriamos visos galimos sistemos trajektorijos, tai leidžia patikrinti, ar sudaryta specifikacija yra teisinga. Teisingumas tiriamas įvairiais validavimo bei verifikavimo (teisingumo tikrinimo ir patvirtinimo) metodais. Funkcionavimo analizės imitacinio modeliavimo priemonėmis metu vykdoma sukurtoji paskirstytosios sistemos specifikacija.

Norint ištirti sistemą, pirmiausia būtina sudaryti agregatinę sistemos specifikaciją. Tam, kad sukurti agregatinę sistemos specifikaciją, pirmiausia reikia gerai išnagrinėti ir aprašyti konceptualųjį sistemos modelį. Konceptualiajam sistemos modeliui aprašyti puikiai tinka universalioji grafinių notacijų kalba UML (angl. *Unified Modeling Language*), skirta įvairių kompiuterinių sistemų analizei, projektavimui ir aprašymui.

Agregatinei sistemos specifikacijai sudaryti naudojamas atkarpomis tiesinių agregatų formalizavimo metodas, aprašytas [4]. Pagrindinis šio metodo pranašumas – galimybė bendrosios formaliosios specifikacijos bazėje patikrinti ar specifikacija sudaryta teisingai, bei sukurti nagrinėjamos sistemos imitacinius modelius. Tai labai svarbu projektuojant sudėtingas sistemas, kadangi reikia ne tik užtikrinti, kad sistemos specifikacija būtų teisinga, bet ir parinkti sistemos parametrus – laikmačiais nustatomas trukmes, informacijos perdavimo kanalų spartą, buferių talpas ir kt.

4.1 KONCEPTUALUSIS PROTOKOLO MODELIS

Detalus funkcijų UDSAccess() ir commit() aprašymas:

- *UDSAccess(t_i, l)* – lokalioji transakcija t_i nuskaito objektus, esančius sąraše l:
    1. Kiekvienam objektui, esančiam sąraše l gaunamas skaitymo užraktas (požymis). Jei bent vienas objektas yra užrakintas (pažymėtas) rašymui, tuomet transakcija užlaikoma kol rašymo užraktas bus atlaisvintas.
- *commit(t_i)* – lokalioji transakcija t_i užbaigė darbą:
    1. Iš UDS gaunamas transakcijos t_i pakeitimų sąrašas WS_t.
    2. Žinutė $<t, WS_t>$ išsiunčiama visiems mazgams ATP priemonėmis.
    3. Kai žinutė su $<t, WS_t>$ pristatoma į bet kurį mazgą:
        a) Kiekvienam objektui o iš WS_t bandoma gauti rašymo užraktą, vykdant šias operacijas vienu nedalomu žingsniu:

i. Jei yra vienas ar daugiau skaitymo užraktų objektui *o*, kiekviena transakcija *t′*, turinti skaitymo užraktą objektui *o*, yra nutraukiama išsiunčiant transakcijos nutraukimo pranešimą $a_{t'}$ ATP priemonėmis ir transakcijai *t* yra suteikiamas rašymo užraktas.

ii. Jei yra rašymo užraktas objektui *o* arba visi skaitymo užraktai objektui *o* yra suteikti transakcijoms t_{recv}, kurių pakeitimų sąrašai $<t_{recv}, WS_{trecv}>$ jau pristatyti, tai transakcijos *t* vykdymas yra atidedamas iki kol šie užraktai bus atlaisvinti.

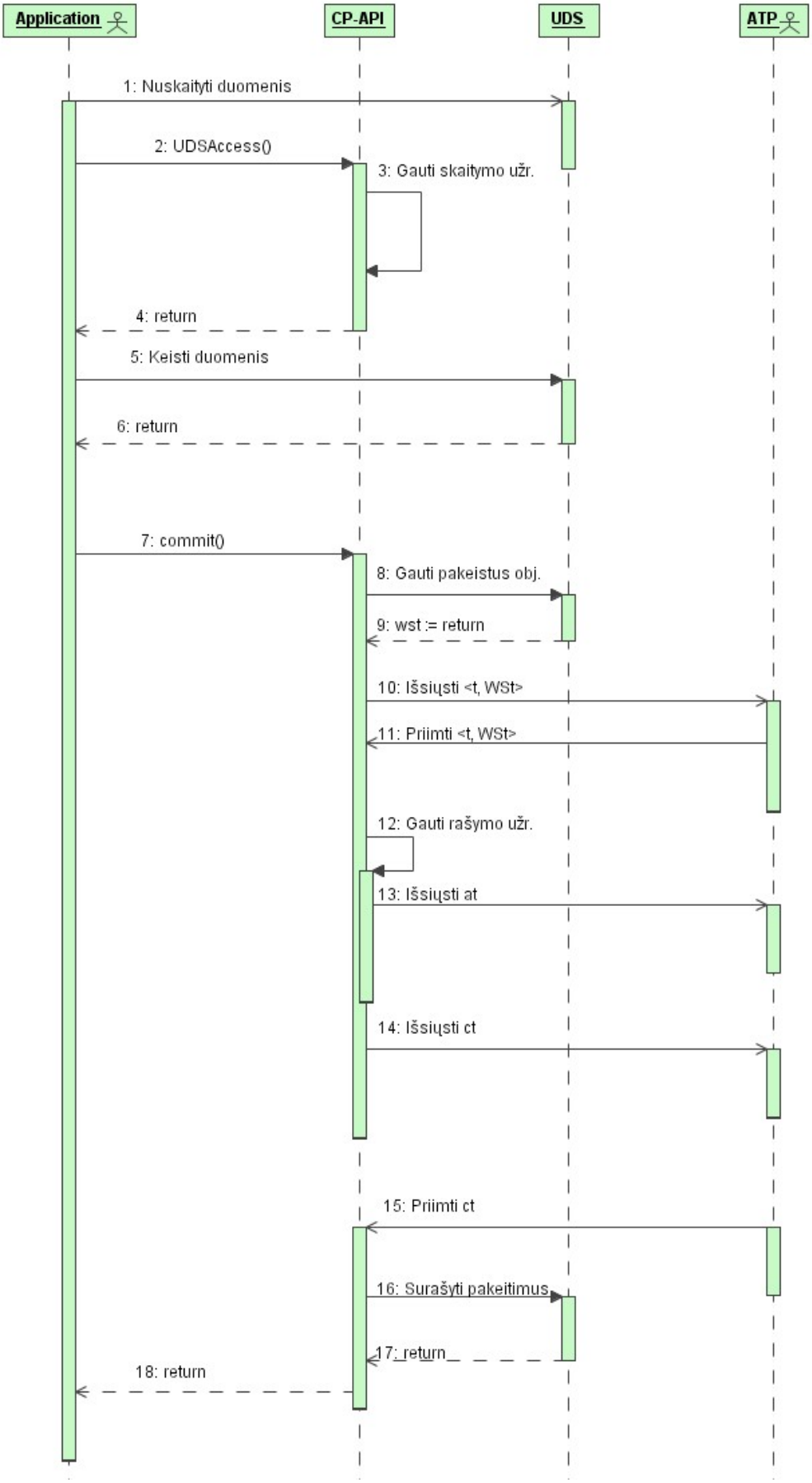
iii. Jei jokių užraktų objektui o nėra, tuomet transakcijai suteikiamas rašymo užraktas.

b) Jei *t* yra lokalioji transakcija, tuomet atominiu transliavimo primityvu išsiunčiamas transakcijos užbaigimo pranešimas c_t.

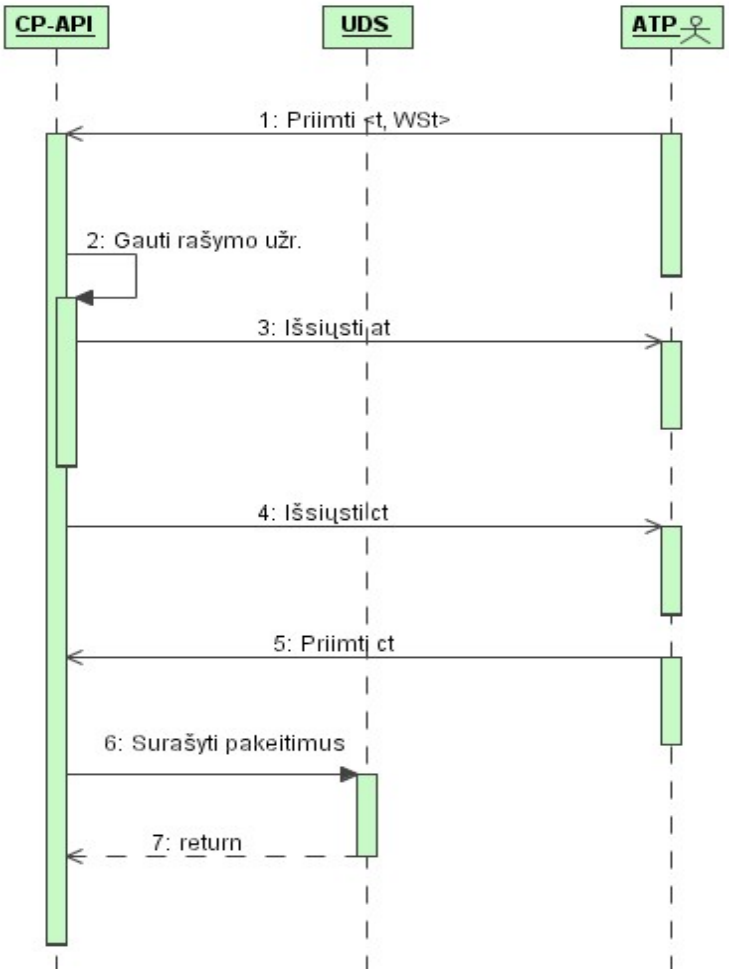
4. Kai pristatomas transakcijos pabaigos pranešimas c_t, visi transakcijos *t* pakeitimai surašomi į duomenų bazę, o visi jos laikomi užraktai – atlaisvinami. Visos transakcijos *t′*, laukiančios rašymo užrakto objektams, kuriuos pakeitė *t*, nutraukiamos išsiunčiant $a_{t'}$ žinutę ATP priemonėmis.
5. Kai pristatomas transakcijos nutraukimo pranešimas a_t, transakcija *t* nutraukiama, o jos laikomi užraktai atlaisvinami.

Iš algoritmo matome, kad visi sprendimai dėl transakcijos priimami deleguojančiame mazge (mazge, kuriame transakcija pradedama): iš šio mazgo išsiunčiamas pakeitimų sąrašas, vėliau šis sąrašas išsiunčia transakcijos pabaigos pranešimą. Konfliktuojančios skaitymo transakcijos nutraukiamos (išsiunčiant nutraukimo pranešimą) taip pat deleguojančiame mazge, kadangi tik deleguojančiame mazge yra skaitymo užraktai, suteikti šioms transakcijoms.

3 ir 5 paveiksluose pavaizduotos sekų diagramos (angl. *sequence diagrams*), parodančios veiksmų išsidėstymą laike. Nuo tada, kai deleguojantis mazgas išsiunčia pakeitimų sąrašą, jis veikia taip pat kaip ir kiti mazgai – t.y. priima sprendimus tik gavęs atitinkamus pranešimus per ATP (išskyrus transakcijos pabaigos sprendimą).



4 pav. Lokaliosios transakcijos veiksmų sekos diagrama



5 pav. Nutolusios transakcijos veiksmų sekos diagrama

Tam, kad nuosekliai išdėstyti konfliktuojančias transakcijas, algoritmas naudoja atominio transliavimo primityvą. Galutinė transakcijų tvarka yra pateikiama pagal <t, WSt> pranešimų eilę. Konfliktų aptikimui naudojama užraktai.

Užraktai saugomi kiekviename mazge nepriklausomai, kitaip tariant kiekvienas mazgas priima sprendimus lokaliai, nepriklausomai nuo kituose mazguose esančių užraktų. Tačiau, kadangi pakeitimų sąrašai ir transakcijų pabaigos pranešimai siunčiami ta pačia tvarka visiems mazgams, visi jie padaro tokius pat sprendimus, nors ir ne sinchroniškai.

Rašymo/rašymo konfliktai iškyla, kai dvi lygiagrečiai vykdomos transakcijos bando keisti tą patį objektą. Tokie konfliktai aptinkami ir sprendžiami naudojant užraktų sistemą: dvi transakcijos bando gauti rašymo užraktą tam pačiam objektui. Kadangi rašymo užraktai yra gaunami po $<t, WS_t>$ gavimo, šių pranešimų gavimo tvarka nustato užraktų gavimo tvarką. Jei transakcija t gaus užraktą rašymui, vėlesnės transakcijos t', bandančios gauti užraktą tam pačiam objektui, vykdymas bus atidėtas. Jei t bus sėkmingai užbaigta, t' bus nutraukta.

Skaitymo/rašymo konfliktai iškyla kai dvi lygiagrečiai vykdomos transakcijos bando prieiti prie to paties objekto: viena – skaitymui, o kita – rašymui. Tokie konfliktai sprendžiami suteikiant pirmumą rašymo transakcijoms. Kai $<t, WSt>$ pranešimas pristatomas, transakcijai t suteikiami rašymo užraktai, tuo pačiu nutraukiant transakcijas, turinčias skaitymo užraktus objektams WS_t sąraše. Ši taisyklė negalioja transakcijoms, kurių rašymo rinkiniai jau buvo pristatyti: šiuo atveju t bus atidėta, kol bus priimtas sprendimas dėl transakcijos, kuri turi skaitymo užraktą.

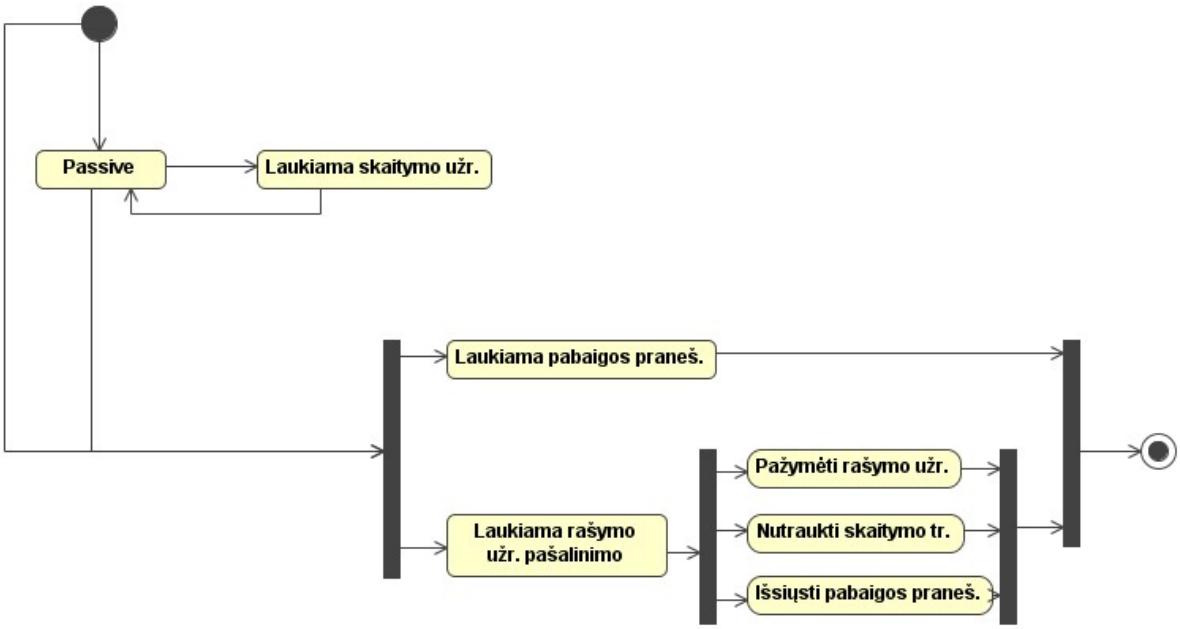
Kelios transakcijos gali užsirakinti tą patį objektą skaitymui. Skaitymo užraktai nėra išskirtiniai užraktai – tai viso labo požymiai, rodantys kurios transakcijos nuskaitė tam tikrus objektus. Jie saugomi tik valdančiajame mazge, kur vyksta transakcija, ir nesiunčiami į kitus mazgus. Kadangi transakcijų daromi pakeitimai išorinėms transakcijoms nematomi iki kol iškviečiama funkcija *commit()*, skaitymo užraktai yra vienintelis būdas sužinoti kokius objektus naudoja transakcija.

Laikomasi principo, kad jei atkeliavęs pakeistų objektų sąrašas $<t, WS_t>$ konfliktuoja su objektais, užrakintais skaitymui, tuomet reikia nutraukti transakcijas, turinčias šiuos užraktus, nes po pakeitimų surašymo lokalioji transakcija veiks naudodama pasenusius duomenis, o ne paskutinę jų versiją.

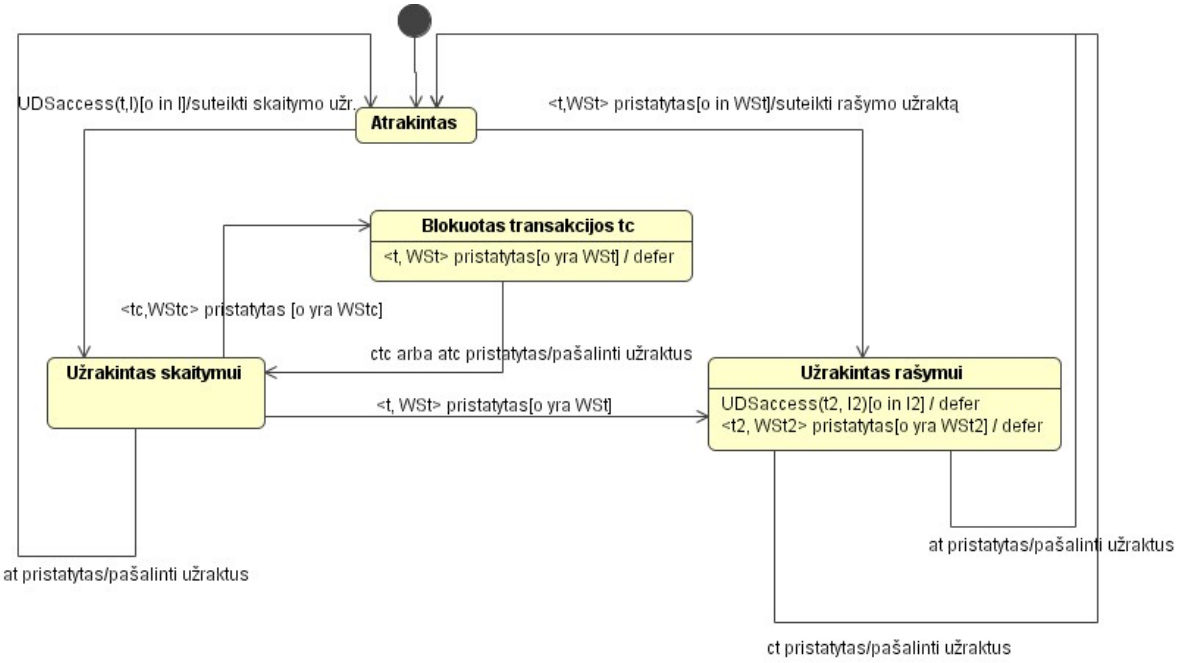
Aprašant formalųjį sistemos modelį, dažnai naudojami būsenų grafai. Šiuo atveju neįmanoma aprašyti sistemos vienu būsenos grafu – pirmiausia dėl to, kad sistema susideda iš kelių lygiagrečiai veikiančių mazgų, kur kiekviename mazge savo ruožtu taip pat veikia lygiagretūs procesai, o antra – dėl to, kad kiekvienas objektas (duomuo) taip pat turi savo būsenas (užraktus).

Galimos transakcijos būsenos pavaizduotos 6 paveiksle. Čia pavaizduota bendra būsenų schema (vykdoma deleguojančiame arba nutolusiuose mazguose), tipiškas transakcijos vykdymo scenarijus. Transakcija gali būti nutraukta nepaisant to, kurioje būsenoje ji yra.

Objekto būsenos transakcijos kontekste pavaizduotos 7 paveiksle. Objektas transakcijai gali būti neužrakintas, užrakintas skaitymui arba užrakintas rašymui. Objektas bus blokuotas transakcijai, jei jis užrakintas rašymui kitos transakcijos.



6 pav. Transakcijos būsenų diagrama



7 pav. Objekto būsenų diagrama transakcijos kontekste

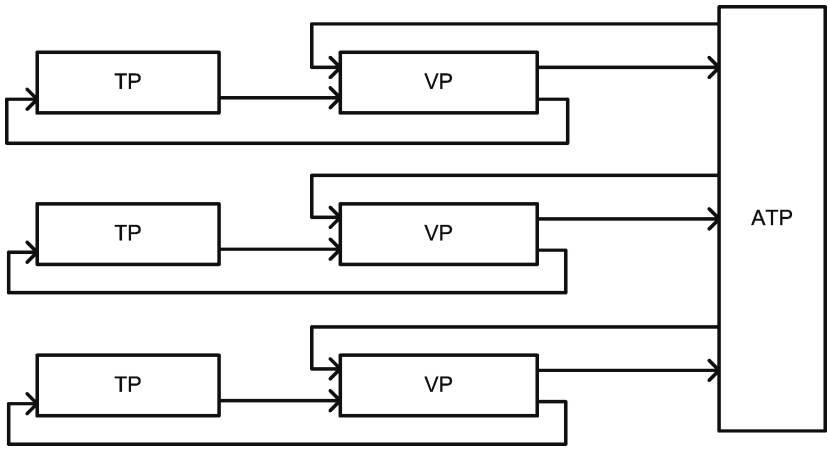
4.2 FORMALUSIS SISTEMOS MODELIS

Agregatų jungimo schema pavaizduota 8 paveiksle. Sistema aprašoma kaip trijų tipų agregatų schema:

1. Taikomųjų programų agregatai TP. Šie agregatai apjungia visas taikomąsias programas, bendraujančias su konkrečiu valdymo mazgu. Agregatas aprašo taikomųjų

programų bendravimą sus sistema CP-API sąsajoje. Kadangi pats programos veikimo principas modeliavimui nėra svarbus, specifikuojami tik jos veikimo dėsniai – kreipinių į CP-API srautai. Schemoje naudojamas vienas agregatas vienam valdančiam mazgui.

2. Vientisumo protokolo agregatai VP. Šie agregatai aprašo pačią vientisumo protokolų logiką, todėl jie ir yra pagrindinis mūsų tyrimo objektas. Schemoje naudojamas vienas agregatas vienam valdančiam mazgui.
3. Atominio transliavimo protokolo agregatas ATP. Šis agregatas aprašo atominio transliavimo protokolo veikimą. Vėlgi, jo funkcionavimo principas modeliavimui nėra svarbus, todėl aprašomas tik jo veikimo dėsnis – protokolo charakteristikos, kuriomis remiantis sukurtas vientisumo protokolas.



8 pav. Agregatų jungimo schema

4.2.1 AGREGATO VP MODELIS

Šis agregatas modeliuoja valdančiojo mazgo darbą (jų sistemoje yra N). Valdančiajame mazgas laikomi dvi aibės – skaitymo užraktų ir rašymo užraktų. Užraktai aprašomi kaip objekto identifikatorius ir transakcijos identifikatoriaus funkcijos. Agregatas su kitais agregatais keičiasi paraiškomis, kurių tipas nustatomas pagal vieną iš įėjimo arba išėjimo parametrų. Transakcijų identifikatoriai ir objektų identifikatoriai nėra apibrėžiami kaip sveiki skaičiai ar kiti konkretūs tipai, viso labo tariama kad jie unikaliai identifikuoja reikiamus resursus.

Kadangi paraiškos apdorojamos eilės tvarka, o tam tikrai atvejais jų apdorojimas atidedamas, užraktų paraiškoms saugoti naudojamos standartinės eilės. Užraktų eilės turi būti peržiūrimos kiekvieną sykį kai transakcija baigiama (nutraukiama arba surašomi pakeitimai), todėl eilių peržiūrai apibrėžti įvykiai, kurie įvyksta tuo pačiu momentu kaip ir transakcijos

pabaigos įvykiai arba paraiškų priėmimo įvykiai.

1. Įėjimo signalų aibė $X=\{x_1, x_2\}$, čia:
   - x_1 = (*TID*, *MSGTYPE*, *OIDSET*) – atėjo paraiška iš taikomosios programos, čia:
     - *TID* – transakcijos identifikatorius;
     - *MSGTYPE* – paraiškos tipas. Galimos reikšmės:
       - *UDSACCESS* – *UDSaccess()* funkcijos iškvietimas;
       - *COMMIT* – *commit()* funkcijos iškvietimas;
       - *ABORT* – *abort()* funkcijos iškvietimas (transakcija nutraukiama);
     - *OIDSET* – objektų identifikatorių sąrašas (aibė), jo interpretacija priklauso nuo paraiškos tipo *MSGTYPE*:
       - kai *MSGTYPE* reikšmė yra *UDSACCESS* – *OIDSET* yra nuskaitomų objektų identifikatorių sąrašas (aibė);
       - kai *MSGTYPE* reikšmė yra *COMMIT* – *OIDSET* yra pakeistų objektų identifikatorių sąrašas (aibė);
   - x_2 – atėjo paraiška iš ATP, čia:
     - *TID* – transakcijos identifikatorius;
     - *MSGTYPE* – paraiškos tipas. Galimos reikšmės:
       - *WRITESET* – atsiųstas pakeistų objektų sąrašas WS_t;
       - *COMMIT* – atsiųstas transakcijos užbaigimo pranešimas c_t;
       - *ABORT* – atsiųstas transakcijos nutraukiamo pranešimas a_t;
     - *OIDSET* – pakeistų objektų identifikatorių sąrašas (aibė), jis perduodamas tik kai *MSGTYPE* reikšmė yra *WRITESET*.
2. Išėjimo signalų aibė $Y=\{y_1, y_2\}$
   - y_1= *(TID, MSGTYPE)* – paraiška taikomajai programai:
     - *TID* – transakcijos identifikatorius;
     - *MSGTYPE* – paraiškos tipas. Galimos reikšmės:
       - *UDSACCESS* – transakcijai suteikti skaitymo užraktai;
       - *COMMIT* – transakcija užbaigta;
       - *ABORT* – transakcija nutraukta;
   - y_2 = (*TID*, *MSGTYPE*, *OIDSET*) – paraiška išsiųsta ATP:
     - *TID* – transakcijos identifikatorius;
     - *MSGTYPE* – paraiškos tipas. Galimos reikšmės:

- *WRITESET* – pakeistų objektų identifikatorių sąrašas <t, WS_t>;
- *COMMIT* – transakcijos užbaigimo žinutė c_t;
- *ABORT* – transakcijos nutraukimo žinutė a_t;

- *OIDSET* – objektų identifikatorių sąrašas (aibė), perduodamas tik kai *MSGTYPE* reikšmė yra *WRITESET*

3. Išorinių įvykių aibė $E' = \{e'_1, e'_2\}$, kur e'_1 - atėjo signalas x_1, e'_2 - atėjo signalas x_2.
4. Vidinių įvykių aibė $E'' = \{e''_{\text{Q\_RL}}, e''_{\text{Q\_WL}}, e''_{\text{rlgrant}}, e''_{\text{wlgrant}}, e''_{\text{commit}}, e''_{\text{send}}\}$,
   - $e''_{\text{Q\_RL}}$ - paraiškų, laukiančių skaitymo užraktų, eilė peržiūrėta;
   - $e''_{\text{Q\_WL}}$ - paraiškų, laukiančių rašymo užraktų, eilė peržiūrėta;
   - $e''_{rlgrant}$ - transakcijai suteikti skaitymo užraktai;
   - $e''_{rlgranted}$ - transakcijai suteikti rašymo užraktai;
   - e''_{commit} - transakcija užbaigta;
   - e''_{send} - paraiška išsiųsta per ATP.
5. Valdymo sekos:

   $e''_{rlgrant} \to \{\xi_{rlgrant}\}, e''_{wlgrant} \to \{\xi_{wlgrant}\}, e''_{commit} \to \{\xi_{commit}\}, e''_{send} \to \{\xi_{send}\}$
6. Tolydžioji agregato būsenos dedamoji:

$$z_v(t_m) = \{w(e'_1, t_m), w(e'_2, t_m), w(e''_{\text{Q\_RL}}, t_m), w(e''_{\text{Q\_WL}}, t_m), w(e''_{\text{rlgrant}}, t_m), w(e''_{\text{wlgrant}}, t_m), w(e''_{\text{wlgrant}}, t_m)\}$$

7. Diskrečioji agregato būsenos dedamoji

$$v(t_m) = \{RL(t_m), WL(t_m), \text{Q\_RL}(t_m), \text{Q\_WL}(t_m), \text{RL\_TID}(t_m), \text{RL\_OIDSET}(t_m), \text{WL\_TID}(t_m), \text{WL\_OIDSET}(t_m), \text{Q\_COMMIT}(t_m), \text{Q\_SEND}(t_m), TLOCAL(t_m)\}$$

   - $RL(t_m)$: *Object_ID* ↔ *Transaction_ID* – skaitymo užraktai;
   - $WL(t_m)$: *Object_ID* ↔ *Transaction_ID* – rašymo užraktai;
   - $Q\_RL(t_m)$ -skaitymo užraktų laukiančių paraiškų eilė;
   - $Q\_WL(t_m)$ – rašymo užraktų laukiančių paraiškų eilė;
   - $RL\_TID(t_m)$ – transakcijos, kuriai suteikiami skaitymo užraktai, identifikatorius;
   - $RL\_OIDSET(t_m)$ – objektų, kuriems suteikiami skaitymo užraktai, identifikatorių aibė;
   - $WL\_TID(t_m)$ – transakcijos, kuriai suteikiami rašymo užraktai, identifikatorius;
   - $WL\_OIDSET(t_m)$ – objektų, kuriems suteikiami rašymo užraktai, identifikatorių aibė;

- *Q_COMMIT(t_m)* – transakcijų, laukiančių užbaigimo, eilė;
- *Q_SEND(t_m)* – paraiškų, laukiančių siuntimo per ATP, eilė;
- *TLOCAL(t_m)* – lokaliųjų transakcijų identifikatorių aibė;

8. Pradinė būsena:
   - $z_v(t_0) = \{\infty, \infty, \infty, \infty, \infty, \infty, \infty\}$
   - $v(t_0) = \{\varnothing, \varnothing, \varnothing, \varnothing, 0, \varnothing, 0, \varnothing, \varnothing, \varnothing, \varnothing\}$
9. Perėjimo ir išėjimo operatoriai:

$\boldsymbol{H}(\boldsymbol{e}_1')$: */Per CP-API gauta paraiška iš taikomosios programos (agregato TP)/*

if (MSGTYPE = UDSACCESS) then

/ Iškviesta funkcija UDSaccess() /

/ Transakcija pažymima kaip lokalioji /

$$TLOCAL(t_{m+1}) = TLOCAL(t_m) \cup \{TID\}$$

/ Skaitymo užraktų paraiška padedama į eilę /

$$ENQ(\text{Q\_RL}(t_m), (TID, OIDSET))$$

/ Artimiausias eilės peržiūros įvykis /

$$w(e''_{\text{Q\_RL}}, t_{m+1}) = \begin{cases} t_m + \xi, & if\ w(e''_{\text{rlgrant}}, t_m) = \infty \\ 0, & otherwise \end{cases}$$

elif (MSGTYPE = COMMIT) then

/ Iškviesta funkcija commit() /

/ WRITESET pranešimas padedamas į eilę išsiųsti per ATP /

$$ENQ(\text{Q\_SEND}(t_m), (TID, WRITESET, OIDSET))$$

/ Artimiausias eilinės paraiškos išsiuntimo per ATP įvykis /

$$w(e''_{send}, t_{m+1}) = \begin{cases} t_m + \xi, & if\ w(e''_{send}, t_m) = \infty \\ w(e''_{send}, t_m), & otherwise \end{cases}$$

elif (MSGTYPE = ABORT) then

/ Taikomoji programa nutraukė transakciją – gali įvykti tik iki iškviečiant commit()/

/ Išvalomi skaitymo užraktai /

$RL(t_{m+1}) = RL(t_m) \setminus (RL(t_m) \rhd \{TID\})$

/ Pašalinama lokalioji transakcijos žymė /

$TLOCAL(t_{m+1}) = TLOCAL(t_m) \setminus \{TID\}$

fi

G(e'$_1$):

y_1 = (TID, ABORT), if MSGTYPE = ABORT

$\boldsymbol{H}(\boldsymbol{e}''_{\mathbf{Q\_RL}})$**:** */Peržiūrėta skaitymo užraktų paraiškų eilė Q_RL /*

/ k - pirmoji transakcija eilėje, nekonfliktuojanti su rašančiom transakcijom /

$$k = \min_{1 \leqslant i \leqslant \#\mathrm{Q\ RL}} \{i \mid \mathrm{Q\_RL}_i(t_m) = (tid, oidset), oidset \cap dom\, WL(t_m) = \emptyset\}$$

if $\exists k$ *then*

/ Jei tokia transakcija egzituoja /

/ Ji pašalinama iš eilės ir pažymima kaip apdorojama /

(RL_TID(t_{m+1}), RL_OIDSET(t_{m+1}) = $Q\_RL_k(t_m)$

DEQ($Q\_RL(t_m)$, k)

/ Jos apdorojimo pabaigos laiko momentas /

$w(e''_{rlgrant}, t_{m+1}) = t_m + \xi$

else

/ Jei tokia neegzistuoja /

$w(e''_{rlgrant}, t_{m+1}) = \infty$

fi

$\boldsymbol{G}(\boldsymbol{e}''_{\mathbf{Q\_RL}})$**:**

Y=$\emptyset$

$\boldsymbol{H}(\boldsymbol{e}''_{\boldsymbol{rlgrant}})$: */Skaitymo užraktai transakcijai suteikti/*

/ Pasižymimi skaitymo užraktai (funkcija Object_ID↔ Transaction_ID) /

$RL(t_{m+1}) = RL(t_m) \cup \{oid \mapsto \mathrm{RL\_TID}(t_m)\}, \forall\, oid \in \mathrm{RL\_OIDSET}(t_m)$

/ Jei dar yra paraiškų eilėje – peržiūrėti eilę /

$w(e''_{\mathrm{Q\_RL}}, t_{m+1}) = t_m$, $jei^{\#}\mathrm{Q\_RL}(t_m) > 0$

$\boldsymbol{G}(\boldsymbol{e}''_{\boldsymbol{rlgranted}})$:

$y_1 = (\mathrm{RL\_TID}(t_m), UDSACCESS)$

$\boldsymbol{G}(\boldsymbol{e}''_{\mathbf{Q\_RL}})$: */ Gauta paraiška iš ATP agregato /*

if (MSGTYPE = WRITESET) then

/ Gautas pakeistų objektų sąrašas /

/ Paraiška padedama į eilę /

$ENQ(\mathrm{Q\_WL}(t_m), (TID, OIDSET))$

/ Eilė peržiūrima, jei nė viena paraiška iš eilės neapdorojama /

$$w(e''_{\mathrm{Q\_WL}}, t_{m+1}) = \begin{cases} t_m, & if\ w(e''_{wlgrant}, t_m) = \infty \\ \infty, & otherwise \end{cases}$$

elif (MSGTYPE = COMMIT) then

/ Gautas transakcijos užbaigimo pranešimas c_t /

/ Paraiška padedama į eilę /

$ENQ(\mathrm{Q\_COMMIT}(t_m), TID)$

/ Eilė peržiūrima, jei nė viena paraiška iš eilės neapdorojama /

$$w(e''_{\mathrm{commit}}, t_{m+1}) = \begin{cases} t_m + \xi, & if\ w(e''_{\mathrm{commit}}, t_m) = \infty \\ w(e''_{\mathrm{commit}}, t_m), & otherwise \end{cases}$$

elif (MSGTYPE = ABORT) then

/ Pristatyta transakcijos nutraukimo paraiška /

/ Išvalomi skaitymo užraktai – iš aibės pašalinami atitinkami ryšiai /

$RL(t_{m+1}) = RL(t_m) \setminus (RL(t_m) \triangleright \{TID\})$

/ Išvalomi rašymo užraktai – iš aibės pašalinami atitinkami ryšiai /

$WL(t_{m+1}) = WL(t_m) \setminus (WL(t_m) \triangleright \{TID\})$

/ Patikrinama ar transakcijos nėra tarp laukiančių rašymo užraktų /

$k = i \,|\, \mathrm{Q\_WL}_i(t_m) = (tid, oidlist),\ tid = TID,\ 1 \leqslant i \leqslant^{\#} \mathrm{Q\_WL}(t_m)$

/ Jei yra – pašalinti iš eilės /

$DEQ(\mathrm{Q\_WL}(t_m), k), \quad if\ \exists k$

/ Jei transakcijos paraiška apdorojam – nutraukti apdorojimą /

$w(e''_{wlgrant}, t_{m+1}) = \infty,\ if\ \mathrm{WL\_TID}(t_m) = TID$

/ Peržiūrėti skaitymo užraktų paraiškų eilę /

$w(e''_{\mathrm{Q\_RL}}, t_{m+1}) = t_m$

/ Peržiūrėti rašymo užraktų paraiškų eilę /

$w(e''_{\mathrm{Q\_WL}}, t_{m+1}) = t_m$

/ Jei transakcija lokalioji - pašalinti žymę iš aibės /

$TLOCAL(t_{m+1}) = TLOCAL(t_m) \setminus \{TID\}$

endif

$\boldsymbol{G(e'_2)}$**:**

/ Jei gauta transakcijos nutraukimo paraiška – TP perduodamas nutraukimo signalas /

$y_1 = (TID, ABORT),\ if\ (MSGTYPE = ABORT) \wedge TID \in TLOCAL(t_m)$

$\boldsymbol{H(e''_{\mathrm{Q\_WL}})}$**:** */Rašančių transakcijų eilės tikrinimo pabaiga/*

/ Suformuojama laikina užrakintų objektų aibė – objektai, užrakinti rašymui bei objektai, užrakinti skaitymui transakcijų, kurių rašymo užraktų paraiškos jau aptarnautos /

$oblockset = (dom\, WL(t_m)) \cup (dom(RL(t_m) \triangleright dom\, WL(t_m)))$

/ Rašymo užraktų paraiškos, kurioms reikia užrakinti bet kurį iš šių objektų, negali būti aptarnaujamos /

/ Randama pirma paraiška eilėje, kuriai nereikia užrakto užrakintiems objektams /

$$k = \min_{1 \leqslant i \leqslant {}^{\#}\mathrm{Q\_WL}} \{ i \mid \mathrm{Q\_WL}_i(t_m) = (tid, oidlist), oidlist \cap oblockset = \emptyset \}$$

if $\exists k$ *then*

/ Jei tokia paraiška egzistuoja /

/ Ji išimama iš eilės ir apdorojama /

$$(\mathrm{WL\_TID}(t_{m+1}), \mathrm{WL\_OIDSET}(t_{m+1})) = \mathrm{Q\_WL}_k(t_m)$$

$$DEQ(\mathrm{Q\_WL}(t_m), k)$$

/ Nustatomas apdorojimo pabaigos įvykio laikas /

$$w(e''_{wlgrant}, t_{m+1}) = t_m + \xi$$

else

$$w(e''_{wlgrant}, t_{m+1}) = infinite$$

fi

$\boldsymbol{G(e''_{wlgrant}):}$

$Y = \varnothing$

$\boldsymbol{H(e''_{wlgrant}):}$ */Apdorota rašymo užraktų paraiška/*

/ Suformuojama laikina transakcijų, kurias reikia nutraukti, aibė – tai transakcijos, kurios turi skaitymo užraktus, konfliktuojančius su rašymo užraktų paraiška/

$$tabortset = ran(RL(t_m) \lhd \mathrm{WL\_OIDSET}(t_m)) \setminus \{\mathrm{WL\_TID}(t_m)\}$$

/ Visoms konfliktuojančioms transakcijoms išsiunčiamas transakcijos nutraukimo pranešimas (pranešimai dedami į siunčiamų pranešimų eilę) /

$$ENQ(\mathrm{Q\_SEND}(t_m), (tid, ABORT, \emptyset)), \forall tid \in tabortset$$

/ Iš užraktų aibės pašalinami skaitymo užraktai, suteikti nutraukiamoms transakcijoms /

$$RL(t_{m+1}) = RL(t_m) \setminus (RL(t_m) \rhd tabortset)$$

/ Į užraktų aibę pridedami rašymo užraktai, suteikti apdorotos paraiškos transakcijai /

$$WL(t_{m+1}) = WL(t_m) \cup \{oid \mapsto \mathrm{WL\_TID}(t_m)\},\ \forall oid \in \mathrm{WL\_OIDSET}(t_m)$$

/ Jei tai lokalioji transakcija – išsiunčiamas transakcijos pabaigos pranešimas c_t /

$$ENQ(\mathrm{Q\_SEND}(t_m), (\mathrm{WL\_TID}(t_m), COMMIT, \phi)), \quad if\ (\mathrm{WL\_TID}(t_m) \in TLOCAL(t_m))$$

/ Nustatomas sekančio pranešimo išsiuntimo laikas /

$$w(e''_{send}, t_m) = \begin{cases} t_m + \xi, & if\ w(e''_{send}, t_m) \neq infinite \wedge {}^{\#}\mathrm{Q\_SEND}(t_{m+1}) > 0 \\ w(e''_{send}, t_m), & otherwise \end{cases}$$

/ Iššaukiamas eilės peržiūrėjimo įvykis /

$$w(e''_{\mathrm{Q\ WL}}, t_{m+1}) = t_m, \quad if\ {}^{\#}\mathrm{Q\_WL}(t_m) > 0$$

G $(e''_{wlgrant})$***:***

$Y = \emptyset$

H (e''_{commit})***:*** */ Transakcijos pabaiga – pakeitimai išsaugoti /*

/Iš eilės pašalinama eilinė transakcijos pabaigos paraiška /

/ tid – pabaigta transakcija /

$$(tid) = \mathrm{Q\_COMMIT}_1(t_m)$$

$$DEQ(\mathrm{Q\_COMMIT}_1(t_m), 1)$$

/ Iš užraktų aibės pašalinami skaitymo užraktai /

$$WL(t_{m+1}) = WL(t_m) \setminus (WL(t_m) \triangleright \{tid\})$$

/ Iš užraktų aibės pašalinami rašymo užraktai /

$$RL(t_{m+1}) = RL(t_m) \setminus (RL(t_m) \triangleright \{tid\})$$

/ Jei tai lokalioji transakcija - pašalinamas lokaliosios transakcijos požymis /

$$TLOCAL(t_{m+1}) = TLOCAL(t_m) \setminus \{tid\}, \quad if\ tid \in TLOCAL(t_m)$$

/ Nustatomas sekantis transakcijos pabaigos momentas /

$$w(e''_{commit}, t_{m+1}) = \begin{cases} t_m + \xi, & if\ {}^{\#}\mathrm{Q\_COMMIT}(t_{m+1}) > 0 \\ \infty, & otherwise \end{cases}$$

G''_{commit}:

/ TP perduodamas transakcijos pabaigos signalas /

$y_1 = (tid, COMMIT), if\ tid \in TLOCAL(t_m)$

$H(e''_{send})$: */ Paraiška išsiųsta per ATP /*

/ Pašalinti eilinę išsiųstą paraišką iš eilės /

$(tid, msgtype, oidset) = \text{Q\_SEND}_1(t_m$

DEQ(Q_SEND(t_m), 1)

/ Nustatomas sekančios paraiškos išsiuntimo įvykio laikas /

$$w(e''_{send}, t_{m+1}) = \begin{cases} t_m + \xi, & if\ ^{\#}\text{Q\_SND}(t_{m+1}) > 0 \\ \infty, & otherwise \end{cases}$$

$G(e''_{send})$:

/ ATP perduodamas paraiškos signalas /

y_2*=(tid, msgtype, oidset)*

4.2.2 AGREGATO ATP MODELIS

Šio agregato tikslas – persiųsti paraiškas, gautas iš valdančiųjų mazgų, visiems grupės valdantiesiems mazgams. Be to jis turi užtikrinti vienodą paraiškų pristatymo eiliškumą visiems grupės mazgams. Tam naudojamas atėjusių paraiškų skaitiklis, kuris didinamas vienetu kiekvieną sykį kai gaunama nauja paraiška. Paraiškai priskiriamas eilės numeris ir ji išsiunčiama kiekvienam mazgui. Siuntimo laikas į skirtingus mazgus gali skirtis, todėl siuntimas modeliuojamas kaip eilė. Keliaudamos paraiškos gali susimaišyti (t.y. paraiška, išsiųsta anksčiau, nukeliaus vėliau nei išsiųsta vėliau), todėl nukeliavusios paraiškos dedamos į eilę iki kol jos bus pristatytos. Paraiška gali būti pristatyta po to, kai pristatomos visos paraiškos, turinčios mažesnį eilės numerį.

1. Įėjimo signalų aibė $X=\{x_1, x_2, ..., x_N\}$, N – mazgų skaičius,
   - x_p = (*TID*, *MSGTYPE*, *OIDSET*) – iš p-tojo mazgo atėjusi paraiška, kurią reikia

persiųsti visiems mazgams. Parametrai *TID, MSGTYPE, OIDSET* neinterpretuojami, o tiesiog perduodami visiems sistemos valdymo mazgams.

2. Išėjimo signalų aibė $Y=\{y_1, y_2,..., y_N\}$, N – mazgų skaičius,
   - y_p = (*TID*, *MSGTYPE*, *OIDSET*) – į *p*-tąjį mazgą pristatyta paraiška.
3. Išorinių įvykių aibė $E^{'}=\{e^{'}_{1,} e^{'}_{2,...}, e^{'}_{N}\}$, kur
   - $e^{'}_{p}$ - x_p signalo atėjimas (iš *p*-tojo mazgo gauta paraiška).
4. Vidinių įvykių aibė $E^{''}=\{e^{''}_{R1}, e^{''}_{R2}, ..., e^{''}_{RN}, e^{''}_{D1}, e^{''}_{D2}, ..., e^{''}_{DN},\}$, kur
   - $e^{''}_{Rp}$, kur $1 \leqslant p \leqslant N$ - paraiška nukeliavo į *p*-tąjį mazgą (paruošta pristatyti);
   - $e^{''}_{Dp}$, kur $1 \leqslant p \leqslant N$ - paraiška pristatyta *p*-tajajam mazgui (nukeliavusios paraiškos pristatomos griežtai pagal eilės numerį gavimo metu).
5. Valdymo sekos: $e^{''}_{Rp} \rightarrow \{\xi_{Rp}\}, e^{''}_{Dp} \rightarrow \{\xi_{D}\}, 1 \leqslant p \leqslant N$, kur:
   - ξ_{Rp} - paraiškos siuntimo į *p* – tąjį mazgą trukmė;
   - ξ_{D} - paraiškos eilės nustatymo (pristatymo) trukmė.
6. Tolydžioji agregato būsenos dedamoji $z_v(t_m)=\{w(e^{''}_{Rp}, t_m), w(e^{''}_{Dp}, t_m)\}, 1 \leqslant p \leqslant N$, kur:
   - $w(e^{''}_{Rp}, t_m)$ - paraiškos atsiuntimo į *p* – tąjį mazgą laiko momentas;
   - $w(e^{''}_{Dp}, t_m)$ - paraiškos pristatymo *p* – tajam mazgui laiko momentas.
7. Diskrečioji agregato būsenos dedamoji
   $v(t_m)=\{NR(t_m), \mathrm{NR\_DLV}_p(t_m) \mathrm{Q\_SND}_p(t_m), \mathrm{Q\_RCV}_p(t_m)\}, 1 \leqslant p \leqslant N$, kur:
   - *NR(t_m)* –agregato gautų paraiškų skaičius laiko momentu t_m;
   - *Q_SND_p(t_m)* – į *p*-tąjį mazgą keliaujančių paraiškų eilė;
   - *Q_RCV_p(t_m)* – į *p*-tąjį mazgą nukeliavusių, bet dar nepristatytų paraiškų eilė.
   - *NR_DLV_p(t_m)* – paskutinės į *p*-tąjį mazgą pristatytos paraiškos numeris
8. Pradinė būsena:

$$w(e^{''}_{Rp}, t_0)=0, \quad 1 \leqslant p \leqslant N$$

$$w(e^{''}_{Dp}, t_0)=0, \quad 1 \leqslant p \leqslant N$$

$$NR(t_0)=0$$

$$\mathrm{Q\_SND}_p(t_0)=emptyset, \quad 1 \leqslant p \leqslant N$$

$$\mathrm{Q\_RCV}_p(t_0)=emptyset, \quad 1 \leqslant p \leqslant N$$

9. Perėjimo ir išėjimo operatoriai:

H $(e^{'}_{p})$***:*** */ Atėjo paraiška iš p-tojo agregato /*

/ Didinamas paraiškos eilės numeris /

$NR(t_{m+1}) = NR(t_m)+1$

/ Paraiška persiunčiama į visus mazgus - padedama į keliaujančių paraiškų eiles. /

$ENQ(\text{Q\_SND}_n(t_m), (t_m+\xi_{Rn}, NR(t_{m+1}), TID, MSGTYPE, OIDSET)), 1 \leqslant n \leqslant N$

/ Artimiausi laiko momentai, kai paraiškos nukeliaus į kiekvieną mazgą /

$$w(e''_{Rn}, t_{m+1}) = \min_{1 \leqslant i \leqslant {}^{\#}\text{Q\_SND}_n(t_{m+1})} \{t \mid \text{Q\_SND}_{n,i}(t_{m+1}) = (t, nr, tid, msgtype, oidset)\}, 1 \leqslant n \leqslant N$$

$\boldsymbol{G(e'_p)}$*:*

$Y = \varnothing$

$\boldsymbol{H(e''_{Rp})}$*: / Į p-tąjį mazgą atkeliavo paraiška /*

/ Atkeliavusios paraiškos numeris keliaujančių paraiškų eilėje /

$$k = i : \text{Q\_SND}_{p,i}(t_m) = (t, nr, tid, msgtype, oidset), t = t_m, 1 \leq i \leq {}^{\#}\text{Q\_SND}(t_m)$$

/ Atkeliavusios paraiškos parametrai /

$$(t, nr, tid, msgtype, oidset) = \text{Q\_SND}_{p,k}(t_m)$$

/ Paraiškos nelieka keliaujančių paraiškų eilėje Q_SND /

$DEQ(\text{Q\_SND}_p(t_m), k)$

/ Paraiška atsiranda tarp laukiančių pristatymo paraiškų eilėje Q_RCV /

$ENQ(\text{Q\_RCV}_p(t_m), (nr, tid, msgtype, oidset))$

/ Sekančios paraiškos atkeliavimo laikas /

$$w(e''_{Rp}, t_{m+1}) = \begin{cases} \infty, if\ {}^{\#}\text{Q\_SND}_p(t_{m+1}) = 0 \\ \min\limits_{1 \leqslant i \leqslant {}^{\#}\text{Q\_SND}_p(t_{m+1})} \{t \mid \text{Q\_SND}_{p,i}(t_{m+1}) = (t, nr, tid, msgtype, oidset)\}, otherwise \end{cases}$$

/Sekančios paraiškos pristatymo laikas. Jei atkeliavusios paraiškos numeris tinkamas, tai ji bus pristatyta /

$$k = i \mid \text{Q\_RCV}_{p,i}(t_m) = (nr, tid, msgtype, oidset), nr = \text{NR\_DLV}_p(t_m)+1, 1 \leq i \leq {}^{\#}\text{Q\_RCV}_p(t_m)$$

$\boldsymbol{G}(\boldsymbol{e}''_{\boldsymbol{Rp}})$**:**

$Y=\emptyset$

$\boldsymbol{H}(\boldsymbol{e}''_{\boldsymbol{Dp}})$**:** */ Pristatyta paraiška p-tajam mazgui /*

/ Eilėje randama paraiška su eilės numeriu, didesniu nei paskutinė pristatyta paraiška /

$$k=i\,|\,\mathrm{Q\_RCV}_{p,i}(t_m)=(nr\,,tid\,,msgtype\,,oidset)\,,nr=\mathrm{NR\_DLV}_p(t_m)+1,1\le i\le^{\#}\mathrm{Q\_RCV}_p(t_m)$$

/ Paraiškos nelieka tarp laukiančių pristatymo paraiškų /

$$(nr\,,tid\,,msgtype\,,oidset)=\mathrm{Q\_RCV}_{p,k}(t_m)$$
$$DEQ(\mathrm{Q\_RCV}_p(t_m),k)$$

/ Paskutinės pristatytos paraiškos numeris /

$$\mathrm{NR\_DLV}_p(t_{m+1})=\mathrm{NR\_DLV}_p(t_m)+1$$

/ Artimiausios laukiančios pristatymo paraiškos numeris /

$$next=\min_{1\le i\le^{\#}\mathrm{Q\_RCV}_p(t_{m+1})}\{t\,|\,\mathrm{Q\_RCV}_{p,i}(t_{m+1})=(nr\,,tid\,,msgtype\,,oidset)\}$$

/ Jei tai – sekanti paraiška, nustatomas jos pristatymo laikas /

$$w(e''_{Dp},t_{m+1})=\begin{cases}t_m+1, if\ \mathrm{NR\_DLV}_p(t_{m+1})+1=next\\ \infty, otherwise\end{cases}$$

$\boldsymbol{G}(\boldsymbol{e}''_{\boldsymbol{Dp}})$**:**

$$y_p=(tid\,,msgtype\,,oidset)$$

4.2.3 AGREGATO TP MODELIS

Šio agregato tikslas – modeliuoti taikomosios programos veikimą. Vienas mazgas aprašomas kaip skirtingų taikomųjų programų paraiškų srautas. Pirmiausia perduodama skaitymo užraktų paraiška, po to – pagal tam tikrą tikimybę perduodama transakcijos pabaigos paraiška (nutraukimas arba pabaigimas surašant pakeitimus). Modeliuojama, kad vienąsyk nuskaitomas atsitiktinis visų sistemos objektų aibės poaibis, o po to – keičiamas šios nuskaitytų objektų aibės poaibis.

1. Įėjimo signalų aibė $X=\{x\}$, *kur*
   - x = (*TID*, *MSGTYPE*) – iš VP gautas pranešimas. Pranešimo tipas nustatomas pagal MSGTYPE. Parametrai:
     - TID – transakcijos identifikatorius;
     - MSGTYPE – pranešimo tipas. Galimos reikšmės:
       - UDSACCESS – UDSaccess() funkcijos pabaiga, transakcijai suteikti skaitymo užraktai;
       - COMMIT – commit() funkcijos pabaiga, transakcija baigta, pakeitimai surašyti
       - ABORT – transakcija nutraukta.
2. Išėjimo signalų aibė $Y=\{y\}$, kur
   - y = (*TID*, *MSGTYPE*, *OIDSET*) – į VP perduodama paraiška. Paraiškos tipas nustatomas pagal MSGTYPE parametro reikšmę. Parametrai:
     - TID – transakcijos identifikatorius;
     - MSGTYPE – pranešimo tipas. Galimos reikšmės:
       - UDSACCESS – iškviesta funkcija UDSaccess() (skaitymo užraktų paraiška);
       - COMMIT – iškviesta funkcija commit() (transakcijos pabaigos paraiška);
       - ABORT – transakcija nutraukta.
     - OIDSET – objektų aibė. Šis parametras perduodamas kai MSGTYPE reikšmė yra UDSACCESS (nuskaitomų objektų aibė) arba COMMIT (pakeistų objektų aibė).
3. Išorinių įvykių aibė , kur
   - e' – iš VP atėjo paraiška (signalas y).
4. Vidinių įvykių aibė $E''=\{e''_{1,tid}, e''_{2,tid}\}$, kur
   - $e''_{1,tid}$ - objektų nuskaitymo įvykis;
   - $e''_{2,tid}$ - transakcijos pabaigos įvykis.
5. Valdymo sekos: $e''_{1,tid}\rightarrow\{\xi_1\}, e''_{2,tid}\rightarrow\{\xi_2\}$
6. Tolydžioji agregato būsenos dedamoji $z_v(t_m)=\{w(e''_{1,tid}, t_m), w(e''_{2,tid}, t_m)\}$
7. Diskrečioji agregato būsenos dedamoji $v(t_m)=\{ROIDSET_{tid}(t_m)\}$
8. Pradinė būsena: $w(e''_{1,tid}, t_0)=\xi_1, w(e''_{2,tid}, t_0)=\infty$, $ROIDSET_{tid}(t_0)=\emptyset$
9. Perėjimo ir išėjimo operatoriai:

$\boldsymbol{H(e'):}$ *- /Gautas pranešimas iš VP /*

if (MSGTYPE = UDSACCESS) then

$w(e''_{2,TID}, t_{m+1}) = t_m + \xi$

else if (*MSGTYPE = COMMIT arba MSGTYPE = ABORT*) *then*

$ROIDSET_{tid}(t_{m+1}) = Ø$

$E'' = \{e''_{1,tid}, e''_{2,tid}\}$

fi

$e''_{1,tid}$

$Y = Ø$

$w(e''_{UDSaccess,tid}, t_{m+1}) = \infty$ */ Taikomoji programa iškvietė UDSaccess funkciją /*

/ oidset – nuskaitomų objektų aibė /

$ROIDSET_{tid}(t_{m+1}) = RND\_SUBSET(OIDSET)$

$w(e''_{UDSaccess,tid}, t_{m+1}) = \infty$

$\boldsymbol{G(e''_{1,tid}):}$

$y = (tid, UDSACCESS, ROIDSET_{tid}(t_{m+1}))$

$\boldsymbol{H(e''_{2,tid}):}$ */ Transakcijos pabaigos momentas /*

$ROIDSET_{tid}(t_{m+1}) = Ø$

$w(e''_{2,TID}, t_{m+1}) = t_m + \xi$

5 IŠVADOS

Duomenų bazių replikavimas tampa vis labiau paplitusi priemonė patikimumui ir našumui didinti. Naujai sukurti paskirstytųjų objektinių duomenų bazių vientisumo protokolai nėra pakankamai ištirti, validuoti ir verifikuoti teoriniame lygmenyje. Todėl šiame darbe aprašytas vieno iš paskirstytųjų objektinių duomenų bazių transakcijų valdymo protokolo formalizavimo procesas yra svarbus ir aktualus norint visapusiškai ištirti šiuos protokolus.

Darbe pasiekti užsibrėžtas tikslas - sukurtas formalusis sistemos modelis. Darbo metu gautos tokios išvados:

- Šis darbas pademonstravo, kad atkarpomis tiesinių agregatų metodas yra tinkamas ir reikalingas analizuojant ir formalizuojant sudėtingus paskirstytųjų sistemų valdymo protokolus. Darbe pademonstuotas formalizavimo procesas, parodantis protokolo analizę ir transformaciją iš žodinio algoritminio protokolo aprašymo į formalųjį protokolo aprašymą.
- Darbe panaudotos UML diagramos yra universali ir patogi priemonė, papildanti konceptualųjį modelio aprašymą. Jos naudojamos kaip pusiau formalus algoritmo aprašymas analizuojant paprastą žodinį algoritmo aprašymą ir siekiant jį paversti formaliuoju modeliu.
- Pastebėta, kad sukurtas protokolo formalusis modelis yra aiškesnis ir vienareikšmiškas, lyginat su žodiniu algoritmo aprašymu, pateikiamu įvairiuose protokolą aprašančiuose dokumentuose. Formalusis modelis yra tinkamas bendravimui tarp algoritmą tyrinėjančių asmenų.
- Sudarytas formalusis protokolo modelis yra tinkamas tęsti protokolo tyrimą teoriniame lygmenyje – validuojant, verfikuojant ar simuliuojant protokolą. Tačiau pastebėta, kad gautasis protokolo formalusis modelis yra gana sudėtingas, todėl pilnam automatizuotam jo tyrimui prireiks daug skaičiavimo resursų.

5.1 TOLESNIS TYRIMAS

Kuriant modelį buvo numatyta, kad ateityje jis bus panaudotas tolesniems tyrimams – protokolo validavimui ir verifikavimui, taip pat jo veikimo simuliacijai. Gautas formalusis modelis yra išties didelis ir sudėtingas, jo galimų būsenų skaičių sunku nuspėti, todėl jo

validavimas ir verfikavimas gali būti nelengvas. Tačiau modelis puikiai tinka simuliacijai, bandomajam vykdymui. Modelio pagrindu galima kurti ir tirti paprastesnius, ne tokius universalius modelius, pavyzdžiui apribojant mazgų, transakcijų ir objektų skaičių.

Tiriant šį modelį, svarbu ištirti ir aplinkybes, kuomet protokolas tinkamas naudoti – COPLA sistema leidžia rinktis protokolus priklausomai nuo reikalavimų našumui ir aplinkos charakteristikų. Kadangi protokolas valdo identiškas duomenų replikas, būtina ištirti kaip keičiasi sistemos našumas didėjant mazgų skaičius, kokią įtaką sistemos apkrovimas daro skaitančių transakcijų nutraukimo koeficientui (angl. *abort rate*).

LITERATŪRA

[1] Powell D. Group communication// Communications of the ACM. ISSN 0001-0782. 1996, Nr. 4, p. 50-53.

[2] Strong replication in the GLOBDATA middleware/ Rodrigues L., Miranda H., Almeida R. ir kt.// Proceedings of the Workshop on Dependable Middleware-Based Systems: tarptautinės konferencijos medžiaga [Vašingtonas, 2002 birželio 26]. Washington, 2002, p. 8-10.

[3] Kemme B., Alonso G. A suite of database replication protocols based on group communication primitives// Proceedings of the 18th International Conference on Distributed Computing Systems (ICDCS): tarptautinės konferencijos pranešimų medžiaga [Amsterdamas, 1998 m. gegužės 26-29 d.]. Washington, 1998, p.156-164.

[4] Pranevičius H.. Kompiuterių tinklų protokolų formalusis specifikavimas ir analizė: agregatinis metodas / Antrasis leidimas. K.: Technologija, 2004, p. 29-36, 42-58.