VILNIAUS UNIVERSITETAS

GEDIMINAS GRICIUS

DAUGIAAGENTINIŲ SISTEMŲ KŪRIMO METODŲ IŠVYSTYMAS NEDIDELIO NAŠUMO ĮTERPTINIŲ SISTEMŲ INTEGRAVIMUI

DAKTARO DISERTACIJA

FIZINIAI MOKSLAI, INFORMATIKA (09 P)

Vilnius, 2015

Disertacija rengta 2010-2014 metais Vilniaus universitete Matematikos ir informatikos institute

Mokslinė vadovė – prof. dr. Dalė Dzemydienė (Vilniaus universitetas, fiziniai mokslai, informatika – 09 P).

Reziumė

Disertacijoje nagrinėjamos daugiaagentinių sistemų taikymo problemos, integruojant į visumą išskirstytas heterogenines įterptines sistemas (mikrovaldiklius, skirtingos paskirties jutiklius, sensorius) belaidžių tinklų aplinkoje. Pagrindinis tyrimo objektas yra programinių agentų sąveiką ir bendravimą užtikrinanti platforma (metodai, programiniai moduliai ir techninės priemonės), kuri leistų integruoti nedidelio našumo įterptines sistemas. Pagrindinis darbo tikslas - išvystyti daugelio programinių agentų sąveikavimo metodus ir tinkamas priemones, kurios užtikrintų nedidelio našumo įterptinių sistemų bendravimą bei bendradarbiavimą, ir pasiūlyti taikymo metodiką ir programines priemones nedidelio našumo įterptinių sistemų integracijai į bendrą dirbtinio intelekto (DI) metodais grindžiamą platformą bei eksperimentiškai išbandyti tokios sistemos veikimo efektyvumą, vykdant adaptyvų meteorologinių parametrų stebėjimą Baltijos jūros akvatorijoje.

Daugiaagentinių sistemų kūrimo metodai leidžia projektuoti sistemas, kurios pasižymi dinamiškumu, lengvu plečiamumu, gebėjimu jungti heterogeninius elementus į visumą, todėl darbe pasirinkta daugiaagentinių sistemų paradigma, tinkama integruoti kuriamas aparatūrines bei programines sistemas. Pastarosios tampa vis labiau išskirstytomis, decentralizuotomis, o jungiant komponentus į bendrą sistemą, kuri siekia bendro tikslo ir gali dirbti belaidžiame tinkle, susiduriama su šiomis problemomis: kaip užtikrinti skirtingų bendravimo protokolų veikimą, kaip integruoti atskirus komponentus, realizuotus skirtingose fizinėse platformose ir skirtingomis programinėmis priemonėmis. Dėl to visos sistemos išmanumas tampa decentralizuotu. Siekiant užtikrinti tokių įterptinių sistemų integravimą į bendrą sistemą, jų sąveiką ir bendradarbiavimą, programinės įrangos lygmenyje bandoma taikyti daugiaagentinių programinių agentų kūrimo metodus ir pasitelkti jų kūrimo metodikas.

Įterptinės sistemos turi fizinių bei techninių apribojimų, kurie kelia tam tikrus reikalavimus kuriamai sistemai. Dažniausiai įterptinių sistemų veikimą apriboja jų kompiuterinis našumas. Šios sistemos turi būti kuo mažesnių gabaritų ir nedidelių energijos išteklių sąnaudų, apsiriboti kuo mažesniais kompiuterinės atminties resursais, nenaudoti didelių operatyviosios atminties kiekių. Tradicinių daugiaagentinių sistemų realizacijai taikomos aukšto lygio interpretuojamos programavimo kalbos (kaip pvz. JAVA), reikalaujančios didelių kompiuterinių pajėgumų. Tačiau tokios kalbos nėra tinkamos mažo našumo įterptinių sistemų darbo užtikrinimui dėl keliamų didelių reikalavimų kompiuteriniams resursams.

Darbe sprendžiami keturi pagrindiniai uždaviniai. Pirmasis uždavinys skirtas daugiaagentinių sistemų veikimo principų ir jų kūrimo metodikų analizei ir jų parinkimui, atsižvelgiant į nedidelio našumo įterptinių sistemų taikymui keliamus reikalavimus. Antrasis uždavinys skirtas daugiaagentinių sistemų kūrimo platformų ir karkasų funkcinei savybių analizei įvertinant tai, kurios iš jų būtų tinkamesnės nedidelio našumo įterptinių sistemų sąveikos užtikrinimui. Trečiasis uždavinys skirtas atlikti eksperimentinį realaus laiko operacinių sistemų įvertinimą ir nustatyti jų tinkamumą daugiaagentinės sistemos realizavimui nedidelio našumo įterptinių sistemų lygmenyje. Ketvirtas uždavinys skirtas sukurti daugiaagentinių sistemų kūrimo metodiką ir realizavimo platformą skirtą nedidelio našumo įterptinių sistemų sąveikos ir komunikacijos užtikrinimui. Penktas uždavinys skirtas atlikti eksperimentinius tyrimus įvertinant jūros hidrometeorologinių duomenų stebėjimui skirtos nedidelio našumo įterptines sistemas integruojančios platformos darbą bei veikimo efektyvumą esant skirtingoms klimato kaitos sąlygų.

Disertaciją sudaro įvadas, penki skyriai, bendrosios išvados, naudotos literatūros bei autoriaus publikacijų disertacijos tema sąrašas.

Įvade formuluojama mokslinio tyrimo problema ir jos mokslinis bei praktinis aktualumas, tyrimų objektas, įvardijamas darbo tikslas bei uždaviniai, aprašoma tyrimo metodika. Pateikiami pagrindiniai mokslinio darbo rezultatai bei disertacijos gautų rezultatų praktinė vertė ir naujumas, disertacijos rezultatų aprobavimas ir sklaida.

Pirmajame skyriuje nagrinėjami daugiaagentinių sistemų veikimo principai, funkcinės galimybės ir keliami reikalavimai intelektualizuotų kompiuterinių sistemų kūrimui. Analizuojamos daugiaagentinių sistemų kūrimo metodikos, ieškoma būdų, kaip pritaikyti jų veikimo principus ir metodus nedidelio našumo įterptinių sistemų veikimo programavimui. Pateikiama dažniausiai naudojamų agentinių sistemų kūrimo metodikų apžvalga.

Antrajame skyriuje nagrinėjamos esamos daugiaagentinių sistemų platformos, jų darbinės aplinkos ir teikiamos funkcinės struktūros, aprašomi svarbiausi šių sistemų komponentai. Nurodomi naudojamų daugiaagentinių sistemų kūrimo platformų trūkumai, kuriuos reikia įvertinti ir pašalinti, norint jas taikyti nedidelio našumo įterptinėse sistemose.

Trečiajame skyriuje nagrinėjamos skirtingos realaus laiko operacinės sistemos ir jų tinkamumas, siekiant užtikrinti nedidelio našumo įterptinių sistemų darbą apribotomis sąlygomis.

Ketvirtajame skyriuje projektuojama DAS platforma ir apibrėžiama jos architektūra, skirta nedidelio našumo įterptinių sistemų kūrimui. Nagrinėjant atskiras įterptines sistemas

kaip DAS agentus, pateikiami jų bendravimo ir sąveikos metodai, tinkami realizuoti belaidžių kompiuterinių tinklų priemonėmis.

Penktajame skyriuje pateikiami eksperimentinių tyrimų rezultatai įvertinant sukurtą hidrometeorologinių stebėjimų Baltijos jūros akvatorijoje sistemą. Pateikiami realizuotos sistemos techninės ir programinės įrangos vertinimo rezultatai, kurie parodo, jog pasiūlyta daugiaagentinės sistemos taikymas palengvina funkcinį plečiamumą ir nedidelio našumo heterogeninių įterptinių sistemų integravimą. Eksperimentiniu būdu pademonstruota kad tokios integruotos įterptinės sistemos yra pajėgios efektyviai dirbti esant ribotam atskirų posistemių darbo našumui.

TURINYS

Paveikslėlių sąrašas

Lentelių sąrašas

Naudojamų sąvokų ir santrumpų sąrašas

AAP (angl. *APRIL Agent Platform*) – APRIL tipo agentinė platforma;

ACL (angl. *Agent Communication Language*) – agentų komunikavimo kalba;

AID (angl. *Agent ID*) – agento identifikatorius;

AMS (angl. *Agent Management System*) – agentų valdymo sistema;

API (angl. *Application Programming Interface*) – taikomųjų programų sąsaja, kurią suteikia operacinė sistema tam, kad taikomoji programa galėtų naudotis viešai pasiekiamų funkcijų, klasių, metodų apibrėžimas, kuriomis galima pasiekti tam tikrą funkcionalumą. Taikomosios programos naudoja OS API (ar kitą API, aprėpiančią šią) tam, kad galėtų valdyti atmintį, failinę sistemą, kitas operacinės sistemos dalis;

AUML (angl. *Agent Unified Modeling Language*) – agentų modeliavimui skirta unifikuota modeliavimo kalba;

BDI (angl. *Believe, Desire, Intention* – BDI) – tikėjimu, troškimu ir ketinimais grindžiami **agentai,** tai tam tikra dirbtinio intelekto metodais grindžiama programinių sistemų kategorija, kuri turi tam tikrus protinio mąstymo bruožus, tokius kaip įsitikinimai arba tikėjimas;

DAS (angl. *multi-agent system*) – daugiaagentinė sistema komponentinio programavimo paradigmos išplėtimas, kurios sudaromos iš tarpusavyje sąveikaujančių agentų. DAS gali būti apibūdintos kaip sujungtas problemos sprendėjų (agentų) tinklas, kurie dirba drauge spręsdami problemą kurios individualiai negali išspręsti;

FIPA (angl. *Foundation for Intelligent Physical Agents*) – Tarptautinė fizinių intelektualizuotų agentų standartų kūrimo organizacija.

HMDRS - Hidrometeorologinių duomenų rinkimo sistema.

HSACS (angl. *Hardware / software artificial cooperative system*) **–** techninės ir programinės įrangos dirbtinė bendradarbiavimo sistema

HSQL (angl. *Hyper Structured Query Language*) – hiper struktūrizuota užklausų kalba;

IIE DAS – intelektuali interaktyvios aplinkos valdymo daugiaagentinė sistema.

ĮS (angl. *embedded systems*) – įterptinės sistemos tai tokia sistema, kurios pagrindinis komponentas yra kompiuterinė techninė įranga su parengta specifine programine įranga, bei turinti specifinę paskirtį;

Įvykio gaišties laikas (ang. *event latency*) – laiko tarpas nuo pertraukimo užklausos pateikimo iki pertraukimo apdorojimo procedūros kodo pirmosios eilutės vykdymo, kai kalbama apie aparatinės įrangos pertraukimus ir laiko tarpas nuo signalinės užduoties sugeneravimo iki užduoties pirmojo nurodymo įvykdymo kai kalbama apie sistemos įvykį;

JADE (angl. *JAVA Agents Development Framework*) – JAVA agentų kūrimo karkasas (platforma);

JADEX (angl. *JAVA Agents Development eXtention Framework*) – JAVA agentų kūrimo platforma, palaikanti BDI tipo agentus;

JADF (angl. *JAVA Agent Development Framework*) – JAVA pagrindu veikiantis agentų kūrimo karkasas;

JAVA – objektiškai orientuota programavimo kalba;

Kompiuterinis našumas yra apibrėžiamas per įtakojančius faktorius tokius kaip centrinio procesorinio įrenginio CPU spartą, atsitinktinės kreipties atminties (RAM) dydžiu, standžiojo disko greičiu ir talpa, kelių uždavinių sprendimu vienu metu.

MIPS (angl. *Millions Instructions Per Second)* – procesoriaus greičio matavimo vienetas nurodantis kiek milijonų instrukcijų jis gali atliekti per vieną sekundę*;*

MTS (angl. *Message Transport System*) – žinučių perdavimo sistema;

Muteksas (angl. *mutex*) – žyma, kuri apsaugo vieną bendrą išteklių, užtikrindama, kad dvi užduotys negalėtų tuo pačiu metu pasinaudoti šiuo ištekliu.

Nedidelio našumo įterptinė sistema (angl. *small scale embedded system*) – įterptinė kompiuterinė sensorinė arba mikrovaldiklio sistema, pasižyminti apribotomis galimybėmis, t.y. ribotu procesoriaus galingumu (1-100 Mips), ribota programų atmintimi (1-256 KB), ribota operatyviąja atmintimi (64-16384 baitų);

OS branduolys (angl. *kernel*) – operacinės sistemos dalis, suteikianti vykdomų programų bei kitų sistemos dalių darbui reikalingus resursus ir funkcijas (atminties valdymas, procesoriaus, virtualios atminties išteklių valdymas, procesų valdymas, aparatinės įrangos abstrakcija, procesų komunikacija ir kt.) kurios pasiekiamos per OS API;

Pirmenybinė inversija (angl. p*riority inversion*) – pirmenybinė inversija atsiranda kai mažesnio prioriteto užduotis užkertą kelią didesnio prioriteto užduočiai ir pakeičia prioritetinę struktūrą;

Planuoklis (angl. *scheduler*) – programinė įranga, atsakinga už gijų valdymą;

Prioritetinis daugelio užduočių atlikimas (angl. *preemptive multitasking*) – kompiuterinėje sistemoje vykdomų procesų valdymo būdas, kai sprendimą perjungti procesorių kito proceso vykdymui priima OS planuotojas, o ne pats aktyvus procesas;

Programinis agentas - autonominė programinės įrangos dalis, vykdanti užprogramuotus situacijos atpažinimo ir valdymo sprendimus ir/arba vartotojo nurodymus. Priklausomai nuo tipų jie gali mokytis, daryti išvadas, tarpusavyje komunikuoti ir veikti siekiant užsibrėžto tikslo;

RTOS (angl. *Real Time Operating System)* - realaus laiko operacinė sistema sąveikaujanti realaus laiko rėžimu, garantuojanti programoms minimalų uždelsimą, kuris gali trukti tarp momento, kai tam tikras prietaisas sugeneravo valdantį signalą iki tol,

kol programa gaus valdymą. Realaus laiko OS naudojamos tais atvejais, kai reikia užtikrinti nepertraukiamą darbą su išoriniais įvykiais;

SACI (angl. *Simple Agent Communication Infrastructure*) – paprasta agentų komunikacijos infrastruktūra;

Sąveikavimas (angl. *interaction*) **–** tai komunikavimą ir bendravimą nusakantis funkcionavimas;

Tinklaveika – tinklaveikos sąvoką apibrėžti, kaip tinklų darbo formavimą tikslingais veiksmais;

UML (angl. *Unified Modeling Language*) – Unifikuota Modeliavimo Kalba;

Vėlinimo svyravimas (ang. *jitter*) – nuokrypis nuo fiksuoto laiko tarpo, skirto užduoties vykdymui, pertraukimo apdorojimui ir t. k.;

XBee – belaidžiai ZigBee tinklo moduliai;

ZCL (angl. *ZigBee Cluster Library*) – ZigBee klasterių biblioteka;

ZigBee – protokolo standartas belaidžiams tinklams.

ĮVADAS

Temos aktualumas

Šiuolaikinės dirbtinio intelekto sistemos (DIS) yra sudėtingos, integruojančios skirtingas, tarpusavyje turinčias komunikuoti programinės bei techninės įrangos komponentes. Norint užtikrinti realiame laike sudėtingomis išorinės sąlygomis dirbančių sistemų veikimą, tenka spręsti heterogeninių komponentų architektūros ir sąveikos klausimus, siūlyti architektūros projektavimo bei sistemos įgyvendinimo metodiką. Šis disertacinis darbas skirtas nedidelio našumo (turinčias ribotą skaičiavimo pajėgumą, atminties talpą ir kt.) heterogeninių įterptinių sistemų integravimui į bendrą integruotos sistemos architektūrą, taikant daugiaagentinių sistemų kūrimo paradigmą.

Įterptinių sistemų klasei priskiriami įvairios paskirties jutikliai ir vykdikliai, sukurti specialių mikrovaldiklių pagrindu ir skirti specializuotų funkcijų vykdymui. Jos gali būti integruojamos į įvairius kompiuterizuotus produktus: prietaisus, įrenginius. Atsiranda vis daugiau šių sistemų taikymo sričių ir sėkmės pavyzdžių: automobilio darbo režimo valdymas, nuotolinis kelių eismo ar statinių būklės stebėjimas, ligonių būklės stebėjimas, pasitelkiant specialiąją medicininę aparatūrą, mechatroninių pavarų ir robotų valdymas.

Dažniausiai įterptinių sistemų veikimą apriboja jų kompiuterinio našumo pajėgumai. Šioms sistemoms keliami reikalavimai: kuo mažesni gabaritai, nedideli energijos ištekliai, kuo mažesni kompiuterinės atminties resursai, nedidelis operatyviosios atminties kiekio naudojimas ir pan. Nedidelio našumo įterptinėmis sistemomis (angl. *small scale embeded systems*) laikomos tokios sistemos, kurios turi ribotą procesoriaus galingumą (1-100 Mips), ribotą programinę (1-256 KB) ir operatyvinę (64-16384 baitų) atmintį. Šias sistemas dažniausiai sudaro mikrovaldiklis (-iai), jutikliai (sensoriai) ar kiti įvedimo įrenginiai, komunikacijos sąsajos bei indikacijos ar išvedimo įrenginys. Jos privalo veikti kuo ilgiau, sunaudodamos minimalų elektros energijos kiekį, ir kartais vykdyti savo funkcijas esant nepalankioms aplinkos sąlygoms. Taip pat pageidautina, kad tokios sistemos dirbtų belaidžio tinklo sąlygomis.

Nagrinėjant įterptinių sistemų, realaus laiko procesų stebėjimui (monitoringui) skirtas technines komponentes, susiduriama su problemomis: šių komponenčių integravimo, jų sąveikavimo užtikrinimas, belaidžių tinklų protokolų parinkimas bei jų darbo užtikrinimas atskirų belaidžio tinklo komponenčių sąveikoje. Svarbu nustatyti, kokie yra tinkami būdai ir metodai, kad būtų galima sujungti išskirstytas heterogenines sistemas į bendrą, kompleksinę,

intelektualiais metodais paremtą sistemą, užtikrinti sistemos veikimą esant savitoms sąlygoms, kai objektyvios aplinkybės neleidžia pasitelkti didelį galingumą turinčias sistemas. Kuriant tokias sistemas tampa labai svarbūs elgsenos aspektai, kuriuos reikia užtikrinti, t. y. jų savarankiškumas, reagavimas į aplinką bei sprendimų priėmimas. Atskiroms įterptinėms sistemoms reikalingas savybes integruojant į bendrą sistemą svarstoma, kaip jos galės bendrauti ir sąveikauti. Daugiaagentinių sistemų kūrimo metodai leidžia projektuoti sistemas, kurios yra dinamiškos, lengvai plečiamos, geba sujungti heterogeninius elementus į visumą, todėl darbe pasirinkta daugiaagentinių sistemų paradigma, kuri pasirodė tinkama kuriamų aparatūrinių ir programinių sistemų integravimui. Aukščiau minėtos sistemos tampa vis labiau išskirstytomis, decentralizuotomis, o komponentus jungiant į bendrą sistemą, kuri siekia bendro tikslo ir veikia belaidžiame tinkle, susiduriama su šiomis problemomis: skirtingų bendravimo protokolų veikimo užtikrinimas, atskirų komponentų, realizuotų skirtingose fizinėse platformose ir skirtingomis programinėmis priemonėmis, integracija.

Atskirų įterptinių sistemų tarpusavio bendravimas ir bendradarbiavimas yra svarbi iki šiol pilnai neišspręsta problema. Šiame darbe siūloma sukurti nedidelio našumo įterptinių sistemų konstravimo metodiką, pasitelkiant agentinių sistemų kūrimo paradigmą. Keliami reikalavimai, jog tokios sistemos būtų savarankiškos, aktyvios ir komunikuojančios, bet tuo pačiu metu nesuvartojančios didelių skaičiavimo ir energijos kiekių. Todėl būtina tikslingai pritaikyti dažnai naudojamą daugiaagentinių sistemų kūrimo metodiką nedidelio našumo įterptinių sistemų įtinklinto bendravimo ir bendradarbiavimo kūrimui.

Siūloma sistemos architektūra, kuri pasižymėtų proaktyvumu, galimybėmis prisitaikyti ir taip padidinti sistemos atsparumą darbui, esant sudėtingoms sąlygoms. Tokių sistemų taikymo sritis galėtų apimti jūros tyrimus ir aplinkos monitoringą, šiuo atveju, sistemos turėtų dirbti sūriame jūriniame vandenyje, esant dideliam banguotumui, vėjo pokyčiams, užšalimui. Be to, daugiaagentinių technologijų taikymas suteiktų galimybes šių sistemų dinaminiam išplėtimui pagal poreikius.

Problemos formulavimas

Programiniai agentai (kaip autonomiškai dirbančios kompiuterinės programos) ir iš jų sudarytos daugiaagentinės sistemos yra sparčiai besivystanti programų inžinerijos ir dirbtinio intelekto šaka. Jų taikymo galimybės gerai žinomos tokiose sistemose kaip e. patarėjai, e. pašto filtrai ir sudėtingesnėse sistemose, sugebančiose valdyti komplikuotas technines sistemas ir net kosminius laivus. Daugiaagentinių sistemų kūrimo metodai leidžia projektuoti sistemas, kurios pasižymi dinamiškumu, lengvu plečiamumu, gebėjimu jungti heterogeninius elementus į

visumą, todėl daugiaagentinų sistemų paradigmos taikymas aparatūrinių ir programinių sistemų integravimui nagrinėjamas daugelyje mokslinių darbų: Carrasco, 2010; Ostaševičiūtė, 2007; Meng 2005; Xing Wu 2009; Tapia 2010; Abidar 2011. Analizuojant ir sprendžiant sudėtingas problemas specialus vaidmuo yra skiriamas agento ir daugiaagentinės sistemos kūrimo etapams (Borodini, 2007). Agentas reiškia abstraktų subjektą, kuris gali išspręsti tam tikrą arba dalinę problemą, keli agentai gali būti integruoti į bendrą sistemą, kurioje veikdami kartu gali susidoroti su sudėtingesnėmis problemomis. Toliau bus laikomasi pagrindinės nuostatos, jog viena įterptinė sistema atitiks vieną agentą. Tradicinių daugiaagentinių sistemų realizacijai taikomos aukšto lygio, interpretuojamos programavimo kalbos (dažniausiai JAVA), reikalaujančios didelių kompiuterinių pajėgumų, tačiau tokios kalbos nėra tinkamos daugiaagentinių sistemų realizavimui nedidelio našumo įterptinių sistemų grupėse dėl keliamų didelių reikalavimų kompiuteriniams resursams.

Integruojant heterogenines nedidelio našumo įterptines sistemas į bendrą sistemą kuri siektų bendro tikslo, susiduriama su šiomis esminėmis problemomis:

- naudojami skirtingi bendravimo ir belaidžio tinklo protokolai,
- posistemės dažniausiai realizuotos skirtingose fizinėse (kompiuterinėse) platformose
- realizuotos skirtingomis loginėmis (programinėmis) priemonėmis.

Pavyzdžiui, jutikliai, atliekantys aplinkos stebėjimo vaidmenį, yra labai diferenciniai dėl vis kitokios stebimų būsenos kintamųjų ir parametrų fizinės prigimties (nuo temperatūros matavimo iki cheminių medžiagų koncentracijų matavimo). Jų duodami signalai yra taip pat skirtingi, todėl jų apdorojimui ir perdavimui reikalinga skirtinga techninė įranga bei programiniai gavimo, analizavimo ir perdavimo metodai. Kiekvienas tokios heterogeninės sistemos įterptinis komponentas gali turėti savo tikslus, tačiau kiekvienas jų taip pat dalyvauja bendroje sistemoje ir prisideda prie bendro sistemos tikslo siekimo. Siekiant užtikrinti heterogeninių sistemų darbą, tenka spręsti resursų išskirstymo uždavinius.

Išskirstytų decentralizuotų sistemų komponentus jungiant į bendrą sistemą, kuri siekia bendro tikslo ir gali dirbti belaidžiame tinkle, susiduriama su šiomis problemomis: kaip užtikrinti skirtingų bendravimo protokolų veikimą, kaip integruoti atskirus komponentus, realizuotus skirtingose fizinėse platformose bei skirtingomis programinėmis priemonėmis (Jamont ir kt., 2013). Šioms sistemoms keliami reikalavimai: sistemos turėtų ir galėtų rinkti informaciją iš supančios aplinkos, veikti autonomiškai, analizuoti situacijos ir aplinkos pasikeitimus, savarankiškai priimti sprendimus ir juos įvykdyti. Todėl jų bendravimui ir

bendradarbiavimui reikalingi efektyvūs belaidžių tinklų protokolai, kurie įgalintų patikimą nedidelio našumo įterptinių sistemų sąveiką. Svarbu šiose sistemose išvystyti ir įvairių situacijų atpažinimo metodus, tačiau dėl per didelės tokių darbų apimties bus apsiribojama tik kai kurių parametrų stebėjimo atpažinimo problemomis.

Heterogeninės įterptinės sistemos pasižymi didesniu techniniu bei programiniu sudėtingumu. Joms reikalinga bendra techninio bei programinio aprūpinimo platforma ir atskiri komponentai, kurie galėtų būti apjungiami tik paskutiniame kūrimo proceso žingsnyje. Šioms sistemoms neretai tenka kurti techniniame lygyje integruotus kodavimo – dekodavimo algoritmus, transformacijų algoritmus, tinklo tvarkykles, duomenų perdavimo protokolus (TCP/IP, ZigBee), kurie padėtų sumažinti procesoriaus apkrovą dėl papildomų skaičiavimų. Visų šių taikomųjų programų veikimą dažniausiai užtikrina realaus laiko operacinės sistemos.

Tyrimo objektas

Metodai ir programinės priemonės, kurios leistų išvystyti daugelio programinių agentų sąveiką ir užtikrinti jų bendravimą taikomą nedidelio našumo įterptinų sistemų tinklaveikoje.

Disertacinio darbo tikslas

Sukurti nedidelio našumo įterptini sistemų bendravimo ir bendradarbiavimo platformą, grindžiamą daugelio programinių agentų sąveikavimo metodais ir priemonėmis, realizuoti ir eksperimentiškai išbandyti jos prototipą, taikant jį Baltijos jūros hidrometeorologinių duomenų stebėsenai.

Tikslui pasiekti keliami šie disertacinio **darbo uždaviniai:**

1. Išnagrinėti daugiaagentinių sistemų veikimo principus, modelius, jų kūrimui skirtas metodikas ir pasiūlyti metodiką, kuri būtų tinkamą nedidelio našumo heterogeninių įterptinių sistemų sąveikai ir integracijai.
2. Ištirti daugiaagentinių sistemų kūrimo platformų funkcines savybes, atlikti jų lyginamąją analizę ir parinkti tas, kurios būtų tinkamos heterogeninių sistemų sąveikai užtikrinti.
3. Išanalizuoti realaus laiko operacines sistemas (RTOS) ir atlikti (greitaveikos) eksperimentinį tyrimą, nustatyti jų tinkamumą daugiaagentinės sistemos realizavimui nedidelio našumo įterptinių sistemų lygmenyje.

4. Sukurti daugiaagentinių sistemų kūrimo metodiką ir realizavimo platformą, skirtą integruoti nedidelio našumo heterogenines įterptines sistemas.
5. Eksperimentiškai verifikuoti pasiūlytą metodiką, realizuojant integruotą sistemos prototipą skirtą jūros hidrometeorologinių duomenų stebėjimui esant skirtingoms klimato kaitos sąlygoms.

Taikomi tyrimų metodai

Analitinėje disertacijos dalyje pateikiama mokslinės literatūros, egzistuojančių daugiaagentinių sistemų kūrimo metodikų ir programinės įrangos analizė. Šioje dalyje taikomi kokybinio tyrimo metodai: informacijos paieška, sisteminimas, lyginamoji analizė bei sukauptos informacijos apibendrinimas.

Projektuojant realaus laiko daugiaagentinės platformos architektūrą, skirtą nedidelio našumo įterptinėms sistemoms, taikytas tyrimo konstravimu metodas (angl. *constructive research*). Šis metodas yra tyrimo procedūra, generuojanti naujoviškas konstrukcijas (modelius, diagramas, metodus, algoritmus ir kt.), skirtas spręsti realaus pasaulio problemoms ir, taikant jas, padaryti tam tikrą įnašą, nagrinėjamos disciplinos teorijoje (Kasanen 1993). Šis tyrimas apėmė realaus laiko OS architektūrų, skirtų nedidelio našumo įterptinėms sistemoms, nagrinėjimą, daugiaagentinės platformos, projektavimą ir kūrimą, kurio metu pasiūlyti nauji metodai bei modeliai, įgalinantys DAS taikymą tokio tipo sistemose.

Siekiant atlikti pasiūlytos programinių agentų architektūros, skirtos nedidelio našumo įterptinėms sistemoms, validaciją buvo atliktas eksperimentinis tyrimas. Tyrimo metu taikant aprašytąją metodiką sukurta integruota daugiaagentinė hidrometeorologinių stebėjimus atliekančių plūdurų sistema. Eksperimentinio tyrimo metu buvo analizuojamos šios sistemos charakteristikos: duomenų perdavimo kiekis ir sistemos adaptacija greitai besikeičiančiose klimato sąlygose. Gauti rezultatai palyginti su šiuo metu naudojamų plūdurų sistemų charakteristikomis.

Gauti darbo rezultatai

Ištirti daugelio agentų sąveikos metodai ir išanalizuotos realaus laiko nedidelio našumo operacinių sistemų taikymo galimybės leido įvertinti, kad realaus laiko operacinė sistema FreeRTOS geriausiai tinka tokių sistemų valdymui, kai yra nedidelis resursų atminties poreikis (kaip pvz., 16-20 MIPS skaičiavimo greičio, 2-8 KB SDRAM tipo atminties, 32-256 KB flash tipo atminties).

Mažo našumo įterptinių valdiklių, kurie veikia kaip atskiri daugiaagentinės sistemos agentai, bendravimui labiausiai tinkami ZigBee Mesh belaidžių tinklų protokolai, kurie buvo įvertinti ir nustatytas jų tinkamumas nedidelio našumo įterptinių sistemų integracijai į bendrą belaidžio tinklo sistemą.

Pasiūlyta daugelio agentų sąveikavimu grindžiamos sistemos išplėtota infrastruktūra ir jos kūrimo metodika, grindžiama DIAMOND metodika ir PROMETHEUS daugiaagentinių sistemų kūrimo platformos principais, kurie yra labiausiai tinkami nedidelio našumo įterptinių sistemų integracijai į bendrą belaidžio tinklo architektūrą.

Eksperimentiškai išbandytas integruotos sistemos veikimas, realizuojant joje daugelio agentų sistemos bendravimo ir bendradarbiavimo savybes. Parodyta, jog pasiūlytos metodikos parinkimas nedidelio našumo įterptinių sistemų integracijai tinka labiau nei kitos parengtos metodikos. Vykdant eksperimentą pademonstruota pasiūlytų ir įgyvendintų sprendimų praktinė reikšmė: parodyta, kad tokio tipo sistema gali būti taikoma sprendžiant ekologinio monitoringo uždavinius, nes veikia esant realioms Baltijos jūros aplinkos ir klimato sąlygoms.

Mokslinis naujumas ir rezultatai

Darbe pasiūlyti metodai ir jų taikymo metodika praplečia daugiaagentinių intelektinių įterptinių sistemų kūrimo galimybes nedidelio našumo įterptinių sistemų integravime. Dauguma siūlomų agentinių sistemų kūrimo platformų yra skirtos didelį pajėgumą turinčių įterptinių sistemų kūrimui, veikiančioms esant stacionarioms sąlygoms ir turinčioms dideles duomenų saugyklas (kompiuterinės atminties). Jų realizacijai dažniausiai taikomos aukšto lygio programavimo kalbos (pvz. JAVA), tačiau jos nėra tinkamos nedidelio našumo įterptinių sistemų darbo užtikrinimui.

Šiame darbe pasiūlyta metodika grindžiama daugelio agentų sąveikos principais ir yra skirta išskirstytų, heterogeninių įterptinių sistemų integravimui. Kiekviena įterptinė sistema kaip atskiras tam tikrų procesų stebėjimui skirtas komponentas, gali turėti savo mikrovaldiklį, specifinių sensorių ir kitų elementų. Įterptinių sistemų integravimas bendroje daugiaagentinių sistemų metodais grindžiamoje sistemoje leido įgyvendinti tikslo siekimo ir sąveikos metodus, pasitelkti ir taikyti sąveikos bei funkcinių agentų galimybes. Manoma, jog gauti moksliniai rezultatai turės įtakos tolimesnėje dirbtinio intelekto sistemų plėtroje, ypač vystant išmaniųjų įrenginių valdymo sistemas.

Sukurta integruota platforma suteikianti galimybes ne tik palengvinti ir pagreitinti daugiaagentinių sistemų fizinę realizaciją, bet ir efektyviai naudoti daugiaagentines

technologijas heterogeninėse įterptinėse sistemose, kas leidžia užtikrinti visų posistemių (agentų-komponentų) tarpusavio suderinamumą ir tokių sistemų praplečiamumą.

Praktinė rezultatų svarba

Mikrovaldiklių taikymo galimybių daugėja daug sparčiau negu jų kaina, todėl įterptinės sistemos integruojamos į prietaisus, mašinas, technologinę įrangą ir vis daugiau lemia jų išmanumą bei išskirtines konkurencines savybes. Darbo rezultatai gali būti sėkmingai panaudoti praktikoje, kuriant ekstremaliomis sąlygomis dirbančias išmaniąsias sistemas.

Praktiškai realizuota ir eksperimentiškai išbandyta daugiaagentinių mažo našumo įterptinių sistemų kūrimo metodika, sukurta technologinė platforma ir atliktas jos eksperimentinis tyrimas bei vertinimas, įrodantis metodikos pagrįstumą.

Pasiūlyta daugelio agentų sąveikavimu grindžiamos sistemos išplėtota infrastruktūra ir jos kūrimo metodika, grindžiama PROMETHEUS daugiaagentinių sistemų kūrimo platformos principais, kurie pasirodė labiausiai tinkami nedidelio našumo įterptinių sistemų integravimui į bendrą belaidžio tinklo architektūrą.

Eksperimentiškai išbandytas integruotos sistemos veikimas, realizuojant joje daugelio agentų sistemos bendravimo ir bendradarbiavimo savybes. Parodyta, kad tokios metodikos parinkimas geriausiai tiko nedidelio našumo įterptinių sistemų integravimui. Taip pat pademonstruota, kad tokio tipo sistema veikia esant realioms Baltijos jūros aplinkos ir klimato sąlygoms. Darbo metu buvo sukurta reali bandymų platforma Klaipėdos universiteto mokslinių bandomųjų tyrimų laboratorijoje. Šiame darbe pateikti rezultatai buvo panaudoti vykdant šiuos MTEP projektus:

- „Įterptinių sistemų mokomųjų laboratorinių stendų kūrimas“ (projekto Nr.: VP2-1.3-ŪM-05-K-02-023), projektas finansuotas pagal Ekonomikos augimo veiksmų programos 1 prioriteto „Ūkio konkurencingumui ir ekonomikos augimui skirti moksliniai tyrimai ir technologinė plėtra“ įgyvendinimo priemonės VP2-1.3-ŪM-05-K „Inočekiai LT“ įgyvendinimą visuotinės dotacijos būdu (2013 -2014 m.).
- „Įmonės klientų informavimo būdų analizė bei šiuolaikiškos ir ekonomiškai adaptyvios informavimo sistemos sukūrimas“ (projekto Nr.: VP2-1.3-ŪM-05-K-03-475), projektas finansuotas pagal Ekonomikos augimo veiksmų programos 1 prioriteto „Ūkio konkurencingumui ir ekonomikos augimui skirti moksliniai tyrimai ir technologinė plėtra“ įgyvendinimo priemonės VP2-1.3-ŪM-05-K „Inočekiai LT“ įgyvendinimą visuotinės dotacijos būdu (2013 -2014 m.).

– „Baltijos Jūros hidrometeorologinių stebėjimo plūdurų duomenų perdavimo sistemų modernizavimas“ projektas, finansuotas pagal Žmogiškųjų išteklių plėtros veiksmų programos 3 prioritetą „Tyrėjų gebėjimų stiprinimas“ įgyvendinimo VP1-3.1-ŠMM-01-V priemonę „Mokslininkų ir kitų tyrėjų mobilumo ir studentų mokslinių darbų skatinimas“ (2014 m.).

Ginamieji teiginiai

1. Mažo našumo įterptinių sistemų bendravimui ir bendradarbiavimui heterogeninėje aplinkoje tikslinga taikyti modifikuotą daugelio programinių agentinių sistemų kūrimo DIAMOND metodiką.
2. Sudėtingose integruotose įterptinėse sistemose, kurias sudaro daug atskirų heterogeninių įterptinių sistemų (mikrovaldiklių), tikslingiau naudoti išskirstytas daugiaagentines sistemas nei vieną centralizuotą agentinę sistemą, esančią viename kompiuteryje, o atskirų komponentų (sistemų) bendravimui naudoti belaidį DigiMesh protokolą.
3. Pasiūlyta metodika ir kūrimo platforma kompleksiškai pritaikyta mažo našumo įterptinių sistemų kūrimui. Eksperimentiškai išbandyta meteorologinių reiškinių stebėjimo sistemai kurti, tai sumažina duomenų perdavimo kiekius, padaro sistemą adaptyvesnę.

Darbo rezultatų aprobavimas

Disertacijos tema yra paskelbti 7 moksliniai straipsniai: 4 recenzuojamuose mokslo žurnaluose, vienas – leidinyje, įtrauktame į Mokslinės informacijos instituto konferencijų darbų sąrašą (ISI Proceedings), 3 kituose respublikinių konferencijų medžiagose (publikacijų sąrašas pateiktas darbo gale).

Disertacinio darbo rezultatai pristatyti 6 tarptautinėse konferencijose ir 2 nacionalinėse konferencijose:

1. International Academic Conference in Social Technologies '13: Social Innovations: Theoretical and Practical Insights, November 10 – 11, 2013, Vilnius.
2. Practical Conference Virtual Instruments in Biomedicine, 2013, Klaipėda, Lithuania.
3. International Workshop, Stochastic Programming for Implementation and Advanced Applications, July 3-6, 2012, Neringa, Lithuania.

4. International Academic Conference Social Technologies '11: ICT for social transformations, November 17 – 18, 2011, Vilnius.
5. 15–oji Respublikinė konferencija Kompiuterininkų dienos (Klaipėda, 2011); rugsėjo 22-24, 2011.
6. The 14th International Conference Electronics‘2010, May 18-20, 2010, Kaunas.
7. The 10th International Conference Reliability and Statistics in Transportation and Communication (RelStat’10), October 20-23, 2010, Riga, Latvia.
8. 7-oji mokslinė konferencija Technologijos mokslo darbai Vakarų Lietuvoje, 2010 m. gegužės 14 d., Klaipėda.

Darbo apimtis ir struktūra

Disertaciją sudaro įvadas, 5 skyriai, išvados, naudotos literatūros sąrašas, autoriaus publikacijų sąrašas.

Disertacijos apimtis - 119 puslapių, iš jų - 10 lentelių ir 35 paveikslėliai.

Įvade aprašomas mokslinio tyrimo aktualumas, analizuojama jo reikšmė, pateikiamas šiuolaikinių mokslinių tyrimų kontekstas ir problemos formuluotė, pagrindžiamas problemos aktualumas, formuluojamas tikslas bei uždaviniai, apibrėžiamos tyrimo hipotezės ir pristatoma tyrimo metodika. Aptariami pagrindiniai disertacinio darbo rezultatai, jų praktinė vertė ir naujumas, disertacijos rezultatų publikavimo rodikliai bei aprobavimas.

Pirmajame skyriuje nagrinėjami daugiaagentinių sistemų veikimo principai, funkcinės galimybės ir keliami reikalavimai intelektualizuotų kompiuterinių sistemų kūrimui. Analizuojamos daugiaagentinių sistemų kūrimo metodikos, ieškoma būdų, kaip pritaikyti jų veikimo principus ir metodus nedidelio našumo įterptinių sistemų veikimo programavimui. Pateikiama dažniausiai naudojamų agentinių sistemų kūrimo metodikų apžvalga. Įvertinus populiarių agentinių sistemų kūrimo metodikų GAIA ir DIAMOND savybes, parodyta, jog tolimesniam darbui labiau tinkama DIAMOND metodika, kuri gali būti taikoma ir nedidelio našumo išskirstytų įterptinių sistemų kūrimui.

Antrajame skyriuje nagrinėjamos esamos daugiaagentinių sistemų platformos, jų darbinės aplinkos ir teikiamos funkcinės struktūros, aprašomi svarbiausi šių sistemų komponentai. Nurodomi naudojamų daugiaagentinių sistemų kūrimo platformų trūkumai, kuriuos reikia įvertinti ir pašalinti, norint jas taikyti nedidelio našumo įterptinėse sistemose.

Trečiajame skyriuje nagrinėjamos skirtingos realaus laiko operacinės sistemos ir jų tinkamumas, siekiant užtikrinti nedidelio našumo įterptinių sistemų darbą apribotomis

sąlygomis. Remiantis atlikta OS savybių analize ir jų našumo eksperimentiniais tyrimais nustatyta, jog tinkamiausia DAS kūrimui yra operacinė sistema FreeRTOS.

Ketvirtajame skyriuje projektuojama DAS platforma ir apibrėžiama jos architektūra, skirta nedidelio našumo įterptinių sistemų kūrimui. Nagrinėjant atskiras įterptines sistemas kaip DAS agentus, pateikiami jų bendravimo ir sąveikos metodai, tinkami realizuoti belaidžių kompiuterinių tinklų priemonėmis.

Penktajame skyriuje pateikiami eksperimentinių tyrimų rezultatai įvertinant sukurtą hidrometeorologinių stebėjimų Baltijos jūros akvatorijoje sistemą. Pateikiami realizuotos sistemos techninės ir programinės įrangos vertinimo rezultatai, kurie parodo, jog pasiūlyta daugiaagentinės sistemos taikymas palengvina funkcinį plečiamumą ir nedidelio našumo heterogeninių įterptinių sistemų integravimą. Eksperimentiniu būdu pademonstruota kad tokios integruotos įterptinės sistemos yra pajėgios efektyviai dirbti esant ribotam atskirų posistemių darbo našumui.

Galutiniame skyriuje pateikiami darbo rezultatai ir išvados, kurios pagrindžia ginamus teiginius.

1. DAUGIAAGENTINIŲ SISTEMŲ VEIKIMO PRINCIPAI IR TAIKYMO GALIMYBĖS

Šiame skyriuje yra nagrinėjami daugiaagentinių sistemų veikimo principai, funkcinės galimybės ir reikalavimai keliami kuriant intelektualizuotas kompiuterines sistemas. Analizuojamos daugiaagentinių sistemų kūrimo metodikos. Ieškoma būdų kaip jų veikimo principus ir metodus pritaikyti nedidelio našumo įterptinių sistemų veikimui programuoti. Pateikiama dažniausiai naudojamų agentinių sistemų kūrimo metodikų apžvalga. Įvertinus gana populiarias agentinių sistemų kūrimo metodikas GAIA ir DIAMOND ir parodyta, jog daugelio agentų sistemoms labiau tinka DIAMOND metodika, kuri gali būti taikoma ir nedidelio našumo išskirstytoms įterptinėms sistemoms kurti.

1.1. Programinio agento ir agentinių sistemų samprata

Programiniai agentai ir iš jų sudarytos daugiaagentinės sistemos yra sparčiai besivystanti programų inžinerijos ir dirbtinio intelekto šaka. **Programinis agentas** gali būti apibrėžiamas kaip autonominė programinės įrangos dalis, vykdanti vartotojo nurodymus arba užprogramuotus situacijos atpažinimo ir valdymo sprendimus (O'Brian, 1998; Nwana, 1996; Wooldridge, 2006). Priklausomai nuo programinių agentų tipų jie gali mokytis, daryti išvadas, komunikuoti tarpusavyje siekdami bendrų tikslų.

Daugiaagentinės sistemos – komponentinio programavimo paradigmos išplėtimas, kurios sudaromos iš tarpusavyje sąveikaujančių agentų. Daugiaagentinės sistemos (DAS) gali būti apibūdintos kaip sujungtų į bendrą visumą problemos sprendėjų-agentų tinklas, kurie dirba spręsdami bendrą, individualiai neišsprendžiamą problemą (Wooldridge, 1995). DAS sistemos yra našesnės ir turi daugiau privalumų, lyginant su vieno agento (monolitinėmis) sistemomis. Jos problemas sprendžia greičiau, išnaudodamos lygiagrečiai galinčius veikti procesus, perduodamos dalinius sprendimus (o ne neapdorotus duomenis), sumažindamos perduodamų duomenų kiekį, taip pasiekiamas didesnis sistemos patikimumas, leidžiant kitiems programiniams agentams perimti neveiksnių agentų pareigas (Moulin, 1996). DAS plačiai naudojamos operatyvinių realaus laiko procesų valdyme ir elektroninių išmaniųjų paslaugų infrastruktūros kūrime.

Savo darbe Wooldridge ir Jennings (1995) programinius agentus apibūdina kaip: *"...techninę įrangą arba (dažniausiai) programos pagrindu sukurtą kompiuterinę sistemą, kuri tenkina šias savybes:*

- *Autonomiškumo: agentas veikia be tiesioginio žmogaus ar ko kito įsikišimo ir kontroliuoja savo veiksmus bei vidinę būseną;*
- *Socialumo: agentas sugeba komunikuoti (palaikyti ryšį) su kitais agentais ir žmogumi tam tikra komunikavimo kalba;*
- *Reaktyvumo (reagavimo): agentas stebi savo aplinką (aplinka gali būti realus pasaulis, grafinė kompiuterio-vartotojo sąsaja, internetas ir kitos, bei jų deriniai) ir laikas tam tikrais laiko intervalais atsako, kokiu tai veiksmu adekvačiai turės įtakos aplinkos pasikeitimui ar įvykusiam įvykiui;*
- *Pro-aktyvumas: agentas imasi veiksmų ne tik tada, kai pakinta aplinka, bet sugeba pats inicijuoti veiksmus aplinkos pakeitimui, jei to reikia jo tikslui pasiekti".*

Brustoloni 1991 m. papildo agento sąvoką teigdamas, jog „*autonominiai agentai – autonomiškai galinčios veikti sistemos, kryptingai veikiančios realiame pasaulyje*". Toks apibrėžimas leidžia išskirti programinį agentą nuo įprastos kompiuterinės programos teigiant, kad programa ir programinis agentas nėra vienas ir tas pats.

Programiniai agentai egzistuoja tam tikroje aplinkoje arba yra jos dalis, kiekvienas agentas jaučia, stebi aplinką ir veikia joje taip, kad darytų įtaką aplinkai jam reikiama kryptimi. Agentai veikia nuolatos tol, kol egzistuoja sąlygos jų veikimui.

Programiniais agentais Brustoloni 2005 metais įvardijo tokias programas, kurios tenkina autonominio agento apibrėžimą, t. y.: *„Autonominis agentas yra sistema, kuri egzistuodama aplinkoje arba būdama jos dalis, vykdo paruoštą planą (užduotį), stebėdama aplinką, ir adekvačiai (atitinkamai) imdamasi priemonių (veiksmų) aplinkos keitimui realizuoja (įgyvendina) planą (užduotį)".*

Autonominiai agentai egzistuoja tam tikroje aplinkoje. Pasikeitus aplinkai, agentas gali nebetekti savo funkcionalumo. Pavyzdžiui, robotas su dienai pritaikytu vaizdo sensoriumi nebegalėtų veikti naktį. Vadinasi, egzistuoja reliatyvumas aplinkos atžvilgiu, t. y., vienos aplinkos atžvilgiu sistema bus agentas, kitos - ne.

Programinių agentų terminas dažnai asocijuojamas su kompiuterine programa, kartais šios sąvokos netgi sutapatinamos. Nors skamba panašiai, tačiau taip nėra. Paprasčiausia kompiuterinė programa veikia realaus pasaulio aplinkoje, t. y., mes į programą suvedame duomenis, o programa, įvykdžiusi algoritmą, grąžina atsakymus, tačiau grąžinti atsakymai neturi jokio poveikio programos veikimui ateityje. Paprasta programa neatitinka ir nuolatinio egzistavimo (veikimo) sąlygos. Vieną kartą ją paleidus, atlikusi užduotį programa baigia darbą,

t. y. gali pradėti veikti tik paleista iš naujo). Apibendrinant galima teigti, kad programinis agentas yra programa, tačiau ne kiekviena programa yra programinis agentas.

Ar sistema yra agentas, ar ne, priklauso ir nuo atliekamų užduočių: egzistuoja reliatyvumas užduoties atžvilgiu. Pavyzdžiui, teksto rinkimui skirtoje programoje realizuota gramatikos tikrinimo-taisymo programa nėra agentas, jei programa elgiasi taip: jai paduodamas tekstas tikrinimui-taisymui, pasileidžia tikrinimo-taisymo algoritmas, gaunamas rezultatas – išvedamas gramatiškai ištaisytas tekstas ir programa baigia darbą. Tačiau minėta tikrinimo-taisymo programa yra agentas, jei būtų realizuota taip: programa spausdinimo momentu nuolatos stebėtų tai, kas rašoma (stebi aplinką), radusi klaidą (pastebėjusi aplinkos pasikeitimą) siūlytų jos taisymo variantus arba pataisytų pati automatiškai, įvertindama rašomo teksto kontekstą. Vadinasi, užduotis, siekiant tikslo, turi būti tokia, kuri tenkintų programinio agento apibrėžimo reikalavimus.

Kiekvienam programiniam agentui daugiau ar mažiau būdingos tam tikros savybės. Jos skirstomos į dvi kategorijas: išorines ir vidines. Išorinėms savybėms galima priskirti komunikavimą, bendro tikslo siekimą, bendradarbiavimą, mobilumą ir pan. Vidinėms savybėms galim priskirti savarankiškumą, mąstymą, mokymąsi, reaktyvumą. Intelektualiųjų programinių agentų savybes nagrinėjo autoriai Wooldridge, 1995; Jennings, 1995; Kinny, 2000; Weerawardhana, 2000 ir kt.

Reaktyvumas - tai savybė, kai sistema geba operatyviai reaguoti į susidariusias situacijas. Reaktyvumas nurodo, jog agentas turi sugebėti atitinkamai reaguoti į aplinką, kurioje gali būti kiti agentai, vartotojai, išoriniai informacijos šaltiniai ar fiziniai objektai. Šis savybės reikalavimas yra vienas iš pagrindinių reikalavimų intelektualiems agentams. Šia savybe pasižymintis agentas privalo turėti tinkamus sensorius, kad galėtų atpažinti ir reaguoti į aplinkos pasikeitimus. Reaktyvumo savybe dažnai pasižymi reaktyvūs (svarstantys, stebėtojai) agentai, kurių pagrindinė užduotis yra stebėti aplinką ir informuoti vartotoją apie įvykusius pasikeitimus.

Iniciatyvumas ir tikslo turėjimas - vienos iš svarbiausių programinio agento savybių. Jeigu agentas ne tik reaguoja į pasikeitimus aplinkoje, bet ir pats imasi iniciatyvos atitinkamomis aplinkybėmis, tai reiškia, kad jis turi iniciatyvumo charakteristiką. Tokiu atveju svarbu, kad agentas turėtų gerai apibrėžtus tikslus ar net sudėtingą tikslų sistemą, leidžiančią gerai atlikti užduotis.

Mąstymo (angl. *reasoning*) ir mokymosi (angl. *learning*) savybės taip pat labai svarbios. Mašininis mokymasis gali būti prižiūrimasis ir mokymasis iš pavyzdžių, tai agentuose realizuotos dirbtinio intelekto savybės. pastarąsias turi agentai, gebantys stebėti savo

aplinką ir, įvykus pasikeitimams, daryti specifines išvadas. Gebėjimas mokytis iš ankstesnių patyrimų ir tinkamai adaptuoti savo veiksmus aplinkoje – svarbi intelektualių agentų charakteristika. Tam būtinas bendravimas su kitais agentais ir galimybė pasiekti informacijos šaltinius.

Savarankiškumas - viena iš pagrindinių savybių, skiriančių agentus nuo tradicinių programų, tai agento gebėjimas siekti savo tikslų savarankiškai be sąveikos su aplinka ar komandų. Kad agentas galėtų veikti savarankiškai, jis privalo turėti savo veiksmų kontrolę, vidinę būseną bei būti aprūpintu šaltiniais ir turi gebėti išspręsti pateiktą užduotį. Tikslų turėjimas ir mąstymas (galimybė mokytis) yra būtini savarankiškumo rekvizitai.

Mobilumas suteikia agentui galimybę judėti/migruoti elektroniniais tinklais iš vieno kompiuterinio tinklo į kitą. Stacionarūs agentai laikomi viename kompiuteryje, nors tokie agentai negali judėti, bet jie gali siųsti pranešimus į kitus kompiuterius ar susisiekti su agentais, esančiais tinkle.

Bendravimo ir bendradarbiavimas savybė pasireiškia tuomet, kai agentas, siekdamas įvykdyti savo užduotis, pasitelkia kitus agentus. Agentas dažnai turi sąveiką su aplinka, kurią sudaro kiti agentai bei vartotojai, norėdamas su jais susisiekti, jis privalo turėti bendravimo galimybę (bendravimo kalbą). Kelių agentų bendradarbiavimas leidžia greičiau ir geriau išspręsti sudėtingas užduotis, su kuriomis nesusidorotų vienas agentas. Kiekvienas agentas turi naudos iš bendradarbiavimo, nes jų tikslai pasiekiami daug greičiau ar net visapusiškai išsprendžiami kitų agentų. Bendradarbiaujantys agentai privalo turėti išplėstą agentų bendradarbiavimo kalbą, nes jiems būtinas ne tik paprastas bendradarbiavimas, bet ir gebėjimas pasidalinti tikslais, privilegijomis, žiniomis ir kt.

Dažnai siekiama, kad agentas turėtų žmogaus charakterio savybių ir gebėtų veikti aplinkoje, imituojant žmogaus elgsenos modelius.

1.2. Programinių agentų tipai ir savybės

Programiniai agentai gali būti skirtingų tipų pagal atliekamas funkcijas ir jiems priskiriamas savybes, t. y. gali būti skirstomi į klases. Agentų tipologija nurodo studijuojamų programinių objektų tipus. Yra keletas bruožų, pagal kuriuos galima klasifikuoti esamus programinius agentus.

Agentai gali būti skirstomi pagal šiuos kriterijus: mobilumą (statiniai arba mobilūs agentai), simbolinio samprotavimo modelį (svarstantys arba reaktyvūs agentai), idealias ir

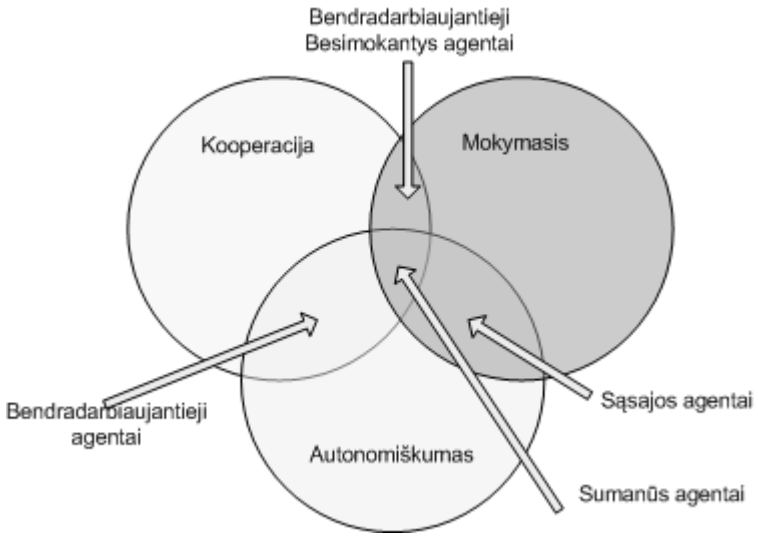
svarbiausias realizuotas savybes (pagal autonomiškumo, kooperacijos, mokymosi atributų buvimą agentai gali būti suskirstyti į bendradarbiaujančius, bendradarbiaujančius besimokančius, sąsajos, sumaniuosius agentus), informacijos valdymo ir manipuliavimo ja vaidmenis (informacijos sistemų arba interneto agentai). Skirtingi agentų tipai ir jų savybės bei šiuos agentų tipus nagrinėjusieji mokslininkai pateikti agentinių sistemų bruožus apibendrinančioje 1 lentelėje.

1. lentelė. Agentų tipų išskirtinės savybės

Agento tipas	**Agentų savybės**		**Darbai, kurie nagrinėja šio tipo agentus**
	Dominuojančios savybės	**Persidengiančios savybės**	
Bendraujantieji agentai	Kooperavimasis, autonomiškumas	Mokymasis	Wooldridge, 1995; Jennings, 1995; Kinny, 2000; Weerawardhana, 2004.
Bendradarbiaujantieji besimokantys agentai	Kooperavimasis, mašininis mokymasis, savarankiškumas	Autonomiškumas	Khojasteh, 2005; Guo-Guan Wang, 2004; Aylett, 1995; Fang-Chang Lin, 1995; Demesure, 2014.
Sąsajos agentai	Mašininis mokymasis, autonomiškumas	Kooperacija	Yan Li, 2008; Kemke, Vasile, 2011.
Sumanūs agentai	Autonomiškumas, iniciatyvumas ir tikslo turėjimas, mokymasis, kooperavimasis, samprotavimas	Mobilumas	Nwana, 1995; Sycara, 1999; Chi, 2006.

Intelektualiųjų programinių agentų klasifikaciją nagrinėjo Nwana, 1996, Wooldridge 1995, DeLoach, 2001, Pranevičius, Raudys, 2008 ir kt. Pagrindinės agentinių sistemų veikimo sritys ir jų persidengimai pateikti 1 paveiksle.

Bendradarbiaujantieji agentai yra autonomiški ir komunikuojantys (su kitais agentais), kad galėtų atlikti savo užduotis. Jie gali mokytis, tačiau šiuo gebėjimu dažniausiai jų darbe nesinaudojama. Formuojant koordinuotą agentų grupę, jiems gali prireikti derėtis, norint gauti abipusius agentų sutikimus tam tikrais klausimais. Pagrindinės šių agentų charakteristikos apima autonomiškumą, socialinius gebėjimus, jautrumą ir iniciatyvumą.



Šaltiniai: (Nwana, 1996; Pranevičius, Raudys, 2008)

1 pav. *Agentinių sistemų savybės ir realizacijose kuriami agentų tipai*

Fig. 1. *Features and implementations of agents types*

Sąsajos agentai dažniausiai apibrėžiami kaip turintys autonomiškumo ir gebėjimų mokytis tam, kad pasiektų jų savininkų tikslus. Sąsajos agentas stebi vartotojo veiksmus, siūlo geresnius būdus jiems atlikti. Taigi sąsajos agentai veikia kaip autonominis asistentas, kuris dirba kartu su vartotoju tam, kad būtų galima atlikti tam tikrus programos darbus. Sąsajos agentai mokosi dėl geresnės pagalbos vartotojui suteikimo.

Mobilieji agentai yra kompiuteriniai programiniai procesai, gebantys „klajoti" kompiuterių tinkluose, sąveikauti su svetimais savininkais, savo savininko vardu rinkti informaciją ir grįžti „namo" atlikus vartotojo duotas užduotis.

Informaciniai agentai buvo sukurti priemonėms, kurios padėtų susitvarkyti su vis didėjančiais informacijos kiekiais arba jų stoka. Tačiau vertėtų pabrėžti, kad skirtumas tarp informacinių, bendradarbiaujančiųjų ir sąsajos agentų yra labai nedidelis: informaciniai agentai apibūdinami pagal tai, ką jie daro, o bendradarbiaujantieji ir sąsajos agentai – pagal tai, kas jie yra.

Reaktyvieji agentai priklauso specifinei grupei, kuri neturi vidinio simbolinio aplinkos modelio, o veikia/atsako veiksmo ir atoveiksmio būdu. Svarbiausia šių agentų savybė ta, kad jie labai paprastai sąveikauja su kitais agentais. Tačiau pažvelgus į agentų grupę globaliai, galima pamatyti sudėtingesnių elgesio modelių.

Sumanieji agentai. Tikslaus jų apibūdinimo nėra, nes negalima tiksliai apibūdinti agento sumanumo. Apskritai, sumanieji agentai šiuo metu yra labiau mokslininkų siekiamybė negu realybė.

Hibridiniai agentai gali turėti kitiems agentams būdingų savybių. Hibridiniai agentai sujungia dviejų ar daugiau minėtų agentų rūšių savybes. Pagrindinė hibridinių agentų atsiradimo priežastis yra ta, kad tam tikrais atvejais reikia turėti vieną skirtingų agentų savybių turintį agentą.

1.3. Autonominių agentų veikimo principai

Autonominis agentas gali būti apibrėžiamas kaip motyvaciją turintis agentas, galintis veikti savarankiškai (Khojasteh, 2005). Autonominiai agentai yra patys save motyvuojantys agentai, t. y., jie patys išsikelia sau tikslus ir jų siekia, užuot laukę kitų agentų nurodymų. Šių agentų elgsena daug sudėtingesnė už neautonominių agentų veikimo galimybes.

Agento veiklos tikslai gali būti generuojami iš motyvacijų, aukštesnio lygio neišvedamų komponentų, nurodančių agentų pobūdį. Motyvai nuo tikslų skiriasi tuo, kad nenurodo siekiamos aplinkos būsenos. Pavyzdžiui, pelno siekio motyvas nenurodo, kokia turėtų būti aplinkos būsena, tačiau gali sąlygoti tikslą, jeigu tai leidžia kiti motyvai (Pranevičius, 2008).

Motyvas yra bet koks noras ar poreikis, kuris gali sąlygoti tikslų formavimą ir paveikti užduotis, atliekamas pasiekti tikslą (Aylett, 1995; Demesure, 2014).

Remiantis autorių Aylett, 1995; Khojasteh, 2005, Demesure, 2014 darbuose apibūdinančiomis autonominio agento funkcinėmis galimybėmis ir išskiriamaisiais bruožais galima teigti, jog tokių agentinių sistemų svarbiausia savybė yra motyvacijos realizavimas ir agento turėjimas veikimo struktūroje bei valdymo algoritmuose.

1.4. Bendradarbiavimo ir bendravimo metodai daugelio agentų sistemose

Agentai retai būna savarankiškomis sistemomis, dažniausiai jie veikia kartu ir sąveikauja vieni su kitais. Sistema, kurią sudaro grupė potencialiai sąveikauti vienas su kitu

galinčių agentų, vadinama daugiaagentine sistema (DeLoach, 1999). Tikslai negali egzistuoti, kol jų nesugeneruoja autonominiai agentai, todėl neįmanoma sistemoje agentams būti be autonomiškumo (Shoham, 1993). Dėl šios priežasties daugiaagentinėje sistemoje turi būti bent vienas autonominis agentas, nors ją gali sudaryti ir daugiau autonominių agentų, tačiau tai nėra būtina sąlyga. Dar vienas reikalavimas agentų aibei, iš kurios kuriama daugiaagentinė sistema, yra tas, kad agentai privalo bendrauti, kitaip jie veiks individualiai. Be to, toks bendravimas turi duoti specialių rezultatų, kurie reikalingi kito agento tikslų įgyvendinimui. Išskiriamas toks daugiaagentinės sistemos apibrėžimas (Pranevičius H., 2008): daugiaagentinę sistemą minimaliai turėtų sudaryti ne mažiau kaip du agentai, iš kurių bent vienas turėtų būti autonominis agentas, ir vienas ryšys tarp dviejų agentų, kurie įvykdo vienas kito tikslus. Jomis dažnai mėginama naudotis net tada, kai kiti problemų sprendimo būdai galėtų būti veiksmingesni.

1.5. Tikėjimais, troškimais ir ketinimais grindžiamų agentinių sistemų veiklos modeliai

Tikėjimais, troškimais ir ketinimais (BDI) grindžiami agentai - tai tam tikra dirbtinio intelekto metodais grindžiama programinių sistemų kategorija, kuri turi protinio mąstymo bruožų: įsitikinimus arba tikėjimą (angl. *Beliefs*), troškimus (angl. *Desires*) ir ketinimus (angl. *Intentions*).

Tokio tipo BDI agento veiklos modelis siejamas su filosofine įsitikinimų-troškimų-ketinimų žmogaus mąstymo realizavimo teorija, kurią pasiūlė Michael Bratman (1985). Šis modelis gali būti išplečiamas panaudojant kitus logikos elementus, pavyzdžiui, Anand Rao ir Michael Georgeff (1995) sistemą.

Integruotais metodais grindžiamos agentinės sistemos BDICTL - tai tikėjimais, troškimais ir ketinimais grindžiami naudojami BDI modeliai, kurių samprotavimai išreikšti daugia-modalinės logikos priemonėmis ir laiko (angl. *temporal logic*) logikos modelio – CTL taikymu (Allen, 1998).

Yra pasiūlytos tokių sistemų realizacijos, kurios leidžia kurti BDI agentus. Viena pirmųjų agentinių sistemų buvo procedūrinio samprotavimo sistema (angl. *Procedural Reasoning System* (PRS)), ją sukūrė komanda, vadovaujama Michael Georgeff. Visos kitos vėliau pasirodžiusios sistemos perėmė PRS modelį.

Wooldridge (1995) išskiria keturias savybes, pagal kurias agentą galima vadinti BDI tipo agentine sistema:

- Išsidėstymas - tai agentų išsidėstymas sistemai priskiriamoje aplinkoje (t. y., neturintys mobilumo savybės);
- Veikimas siekiant tikslo;
- Reaktyvumas, kai agentai reaguoja į aplinkos pokyčius;
- Bendravimo savybė, t. y., bendraujantys agentai, kai jie gali bendrauti su kitais agentais, aplinka bei kitais vartotojais.

Agentų įsitikinimai parodo agento informacinę būseną, kitaip tariant, parodo jo žinias apie supantį pasaulį, kitus agentus bei jį patį. Įsitikinimai gali būti išreiškiami ir išvadomis, kurios, vykstant tiesioginiam išvedimui (angl. *forward chaining*), priveda prie naujų įsitikinimų. Dažniausiai visa ši informacija saugoma duomenų bazėje.

Naudojant terminą įsitikinimai, o ne žinios pabrėžiama, kad ne viskas, į ką agentas tiki, yra būtinai teisinga (ateityje faktai gali keistis).

Troškimai parodo agento stimulą. Jie atstovauja situacijas arba agento siekiamus tikslus, pavyzdžiui, surasti geriausią prekės kainą, sunaikinti priešininką ir pan.

Ketinimai atstovauja svarstymo būseną: ką agentas privalo daryti. Taigi ketinimai yra tokie norai, kuriuos agentas labiausiai nori įgyvendinti.

Planai - tai veiksmų sekos, kurias įvykdęs agentas įgyvendins bent vieną iš savo ketinimų.

Ne visos apžvelgtos sistemos naudoja agentiškai orientuoto programavimo priemones, dažniausiai pasitelkiama scenarijų kalba, kuri praktikuojama robotų valdyme. BDI tipo agentų problemos (agentai – chameleonai arba tarpusavyje bendraujantys ir duomenimis besikeičiantys agentai) analizuojami darbuose.

1.6. Įterptinių sistemų samprata

Wayne Wolf, pateikdama įterptinės sistemos apibrėžimą, teigia, kad bet kuris įrenginys, turintis savyje programuojamą kompiuterį (pvz. mikrovaldiklį), bet nesantis bendrosios paskirties kompiuteriu, gali būti vadinamas įterptine kompiuterine sistema arba tiesiog įterptine sistema (Wolf, 2008). Tad galima teigti, jog įterptinė sistema - tai tokia sistema, kurios pagrindinis komponentas yra kompiuterinė techninė įranga su parengta programine įranga bei specifine paskirtimi.

Įterptinių sistemų klasifikavimas

Įterptinės sistemos dažniausiai klasifikuojamos į tris grupes: nedidelio našumo (angl. *small scale*), vidutinio našumo (angl. *medium scale*) ir sudėtingos įterptinės sistemos (ang. *sophisticated embaded systems*) (Kamal R., 2009).

Nedidelio našumo įterptinės sistemos kuriamos panaudojant vieną 8 arba 16 bitų mikrovaldiklį, dažniausiai jos turi nesudėtingą programinę įrangą ir nedaug papildomos techninės įrangos. Programuojant šio tipo sistemas, neretai naudojama asemblerio arba „C“ programavimo kalba, kuria parašytas programinis kodas kompiliuojamas į vykdomąjį kodą ir patalpinamas mikrovaldiklio atmintyje. Programinė įranga turi tilpti į ribotą mikrovaldiklio atmintį bei užtikrinti visos sistemos sklandų vykdymą.

Vidutinių apimčių įterptinės sistemos dažniausiai kuriamos panaudojant vieną ar kelis 16 arba 32 bitų mikrovaldiklius bei kitą papildomą įrangą DSP, RISC. Į šias sistemas taip pat gali įeiti pridėtiniai numatytos paskirties procesoriai (angl. single prupose processors), skirti įvairių sistemos funkcijų vykdymui (pavyzdžiui, duomenų perdavimo protokolams, įvairių jutiklių nuskaitymui ir skaitmeninimui). Šio tipo sistemos turi sudėtingesnę techninę įrangą, todėl kuriant programinę įrangą neretai naudojamos C/C++/JAVA programavimo kalbos bei realaus laiko operacinės sistemos (RTOS).

Mažo našumo įterptinės sistemos pasižymi šiomis savybėmis: ribotu procesoriaus galingumu (1-100 Mips), ribota programų atmintimi (1-256 KB), ribota operatyviąja atmintimi (64-16384 baitų). Pagal šias savybes tokiose sistemose galimi keli programinės įrangos realizavimo būdai: pagrindinis ciklas su pertraukomis, realaus laiko operacinės sistemos (RTOS).

1.7. Agentinių sistemų kūrimo metodikos ir jų palyginimas

Apžvelgsime keletą agentinių sistemų kūrimo metodikų, kurios siūlo agentinių sistemų kūrimo ir projektavimo būdus. Renkantis metodiką, svarbu įvertinti tiek kuriamo objekto esmines savybes, tiek ir metodikos svarbiausias ypatybes. Kadangi kiekviena sistema turi jai keliamų specifinių reikalavimų rinkinį, jie ir nurodo, kurie vertinimo kriterijai yra svarbiausi ir turi būti užtikrinti kūrimo procese.

Įterptinės sistemos turinčios techninę įrangą, kuri nėra susijusi vien programiniais komponentais, apima ir aplinką, kurioje jie veikia. Atsižvelgiant į tai, šiose sistemose būtinas programinės įrangos elementų ir aparatūrinės (jutiklių, vykdiklių) dalies integravimas.

Metodikos, skirtos agentinėms sistemoms kurti, turėtų susidėti iš šių etapų: analizės, architektūros projektavimo, detalaus projektavimo, realizacijos, o taip pat ir veikimo bei palaikymo, įskaitant ir galimybę esant reikalui perprojektuoti sistemą veikimo metu. Kartu, tokia metodika privalo palaikyti realaus laiko bei patikimumo reikalavimų specifikacijas. Daugiaagentinių sistemų kūrimo metodikų palyginimas bus vykdomas atsižvelgiant į žemiau pateikiamus kriterijus.

Kūrimo gyvavimo ciklo modelis. Skirtingos daugiaagentinių sistemų kūrimo metodikos gali naudoti skirtingus kūrimo gyvavimo ciklo modelius (tokius kaip iteracinis, inkrementinis, spiralinis, prototipavimo ir k.t.)

Kokius kūrimo etapus apima gyvavimo ciklas. Šis kriterijus nurodys kokius sistemos kūrimo etapus apima daugiaagentinių sistemų kūrimo metodika (pvz. analizė, projektavimas, realizavimas).

Kokio tipo sistemoms kurti skirta. Šis kriterijus nurodo kokio tipo sistemoms kurtis skirta metodika. Metodikos, skirtos specifinėms dalykinėms sritims, nesusijusioms su įterptinėmis sistemomis (pvz. skirtos modeliavimui) nenagrinėjamos.

Kuriamos sistemos dydis. Šis kriterijus nurodo kokio dydžio sistemoms kurti skirta daugiaagentinių sistemų kūrimo metodika. Reikalavimas kuriamos sistemos dydžiui yra daugiau rekomendacinio pobūdžio.

Agentų tipai. Šis kriterijus analizuoja, ar sistemą siūloma kurti iš vieno tipo, ar skirtingų agentų tipų.

Modeliavimo įrankiai. Projektuojant agentines sistemas pravartu turėti modeliavimo įrankius grindžiamus pasirinktąja daugiaagentinės sistemos kūrimo metodika. Šis kriterijus nurodo ar atitinkamai kūrimo metodikai yra sukurti modeliavimo įrankiai.

Tikslų specifikavimas. Šis kriterijus nurodo ar metodika leidžia specifikuoti sistemos ir agentų tikslus.

Rolių specifikavimas. Šis kriterijus nurodo ar metodika leidžia specifikuoti roles (vaidmenis). Rolių specifikavimas sumažina sistemos sudėtingumą dekomponuojant ją į mažesnes dalis.

Bendravimo protokolų specifikavimas. Šis kriterijus nurodo ar metodika numato bendravimo protokolų specifikavimą. Agentų komunikacijos palaikymas yra labai svarbus visose agentinėse metodikose, kadangi komunikacija yra viena iš esminių agentų charakteristikų.

Agentų-aplinkos bendravimo specifikavimas. Agentai yra išdėstyti aplinkoje, kurią jie jaučia ir veikia. Šis kriterijus nurodo ar metodika numato agentų ir aplinkos komponentų (jutiklių, vykdiklių) bendravimo specifikavimą.

Sistemos architektūros specifikavimas. Šis kriterijus nurodo ar metodika leidžia modeliuoti visą sistemą kaip organizaciją, analogišką žmoniškosioms organizacijoms, kuriose visi žmonės elgiasi savarankiškai, tačiau laikydamiesi iš anksto nustatytų taisyklių.

Aparatūros keliamų ribojimų specifikavimas. Šis kriterijus nurodo ar metodika leidžia specifikuoti ribojimus, kurie bus keliami techninei įrangai.

Laiko reikalavimų specifikavimas. Šis kriterijus nurodo ar metodika numato laiko reikalavimų specifikavimą, ar leidžia užduotims specifikuoti laiko reikalavimo tipus : fiksuotas laikas, laiko intervalai, galutiniai terminai bei pradžios laikai.

Įvykių prioritetų specifikavimas. Šis kriterijus nurodo ar metodika numato skirtingų įvykių prioritetų specifikavimą. Prioritetų priskyrimas įvykiams yra vienas iš būdų palengvinančių sudaryti pranešimų apdorojimo ir perdavimo sekas.

1.7.1. GAIA agentinių sistemų projektavimo metodika

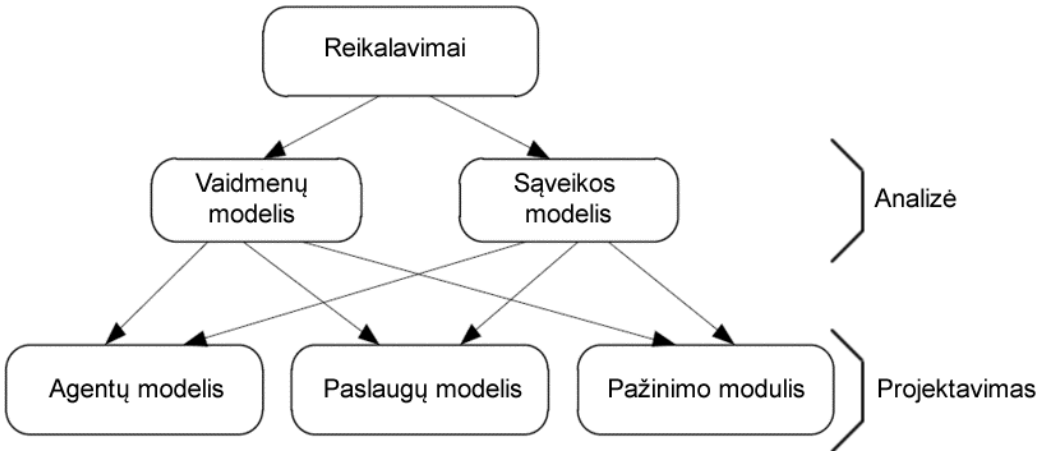
GAIA agentinių sistemų projektavimo metodika buvo pasiūlyta (Wooldridge, Jennings, Kinny, 2000). Ji apibrėžia sistemą, kaip dirbtinę struktūrą, turinčią heterogeninius agentus, kurie bendrauja vienas su kitu tam, kad pasiektų bendrus tikslus, kurie susideda iš tam tikrų atskirų agentų sub-tikslų. GAIA metodika, skirta daugelio agentų sistemų analizei ir projektavimui. Ši metodika yra tiek bendra, kadangi ji tinkama plačiam daugiaagentinių sistemų spektrui, tačiau ir išsami, kadangi apima sistemos aspektų makrosluoksnį (agentų organizacijas) ir mikrosluoksnį (projektuojant atskirą agentą). Apibendrinti GAIA metodikos modeliai yra pateikti 2 pav.

GAIA metodikos pagrindas yra daugiaagentės kompiuterinės sistemos atliekančios įvairius sąveikaujančius vaidmenis.

Analizė GAIA metodikoje, prasideda abstrakčiomis ir baigiasi pakankamai konkrečiomis sąvokomis.

Analizės etapo tikslas - sukurti sistemos sąveikos ir vaidmenų pasiskirstymo supratimą ir jos struktūrą (nenurodant jokių įgyvendinimo detalių). Formuojamas sistemos veikimo supratimas, kuris yra užfiksuojamas sistemos organizavime. Sistemos organizacija tai vaidmenų rinkinys, kuris išreiškia kaip šie vaidmenys vienas su kitu sąveikaus (nustatomos

sąveikos vaidmenys tam tikrame santykyje) ir kaip jie dalyvaus sisteminiuose, paplitusiuose sąveikų modeliuose su kitais vaidmenimis.



Šaltinis: sudaryta pagal (Wooldridge, Jennings, Kinny, 2000)

2 pav. *Ryšiai tarp GAIA modelių*

Fig. 2. *Relations between GAIA models*

Pagrindiniai GAIA metodikoje naudojamos esybės gali būti suskirstytos į dvi kategorijas: abstrakčios ir konkrečios. Abstrakčios esybės yra tos, kurios naudojamos analizės metu sistemos konceptualizavimui, bet kurios nebūtinai turi tiesioginį realizavimą sistemoje. Metodikoje abstrakčios esybės yra: vaidmenys, pareigos, leidimai, veiklos, protokolai, gyvybingumas, saugumas. Konkrečios esybės priešingai yra naudojamos projektavimo metu ir paprastai turės realizacines kopijas kuriamojoje sistemoje. GAIA metodikoje konkrečios esybės yra: agento tipai, paslaugos, pažinimas. Pati abstrakčiausia sąvoka sąvokų hierarchijoje yra sistema.

Pagal agentinių sistemų kūrimo paradigma, kuriama sistema yra suprantama kaip dirbtinė visuomenė arba tam tikrus vaidmenis atliekančių objektų organizacija.

Sistemos veiklos idėja ją traktuoti kaip veikiančią visuomenę yra naudinga, galvojant apie sekantį lygį hierarchijoje, t. y. priskiriamus ir vykdomus vaidmenis. Kompiuterinę sistemą, apibūdina eilė vaidmenų, tačiau ši idėja gana natūrali, taikant organizacinio pasaulio vaizdą (sistemos veikimo idėją).

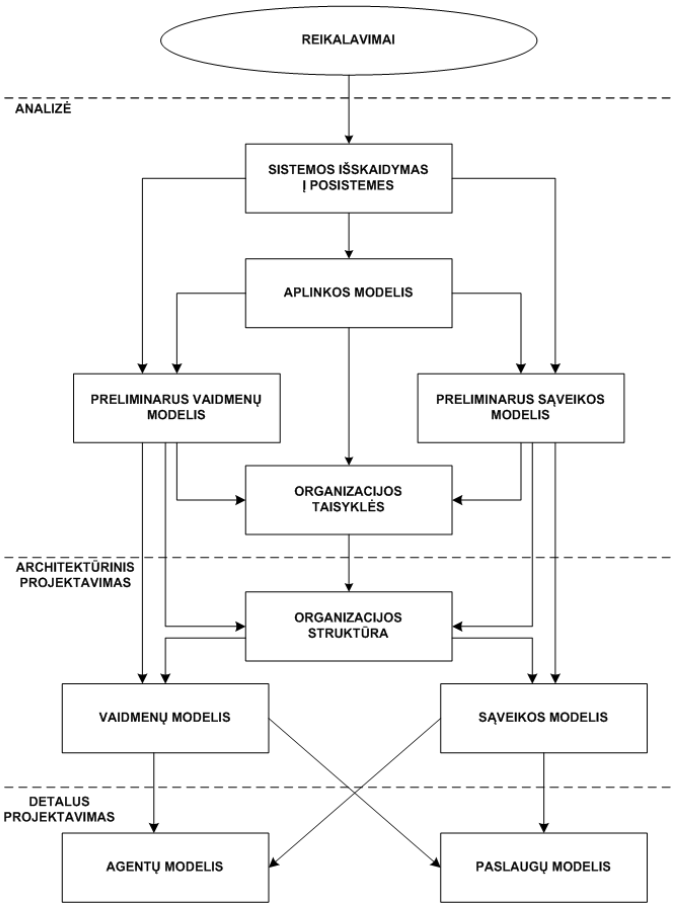
Vaidmuo yra apibrėžiamas keturiais požymiais:

- pagal pareigas,
- pagal priskiriamus leidimus,
- pagal veiklos modelius ir
- vykdomus protokolus.

Pareigos reiškia funkcionalumą ir greičiausiai yra pagrindinis požymis, susietas su vaidmeniu. Kaip pavyzdys, pareigų, susijusių su kompanijos prezidento vaidmeniu, galėtų būti kasmetinis akcininkų sušaukimas. Pareigos gali būti padalintos į du tipus: gyvybingumo savybės ir saugumo savybės pagal (Wooldridge, Jennings, Kinny, 2000).

Gyvybingumo savybės realizacija užtikrina nepertraukiama sistemos veikimą. Jos apibūdina tas veiklos būkles, kurias agentas turi sukelti, kai duotos tam tikros aplinkos įvertinimo sąlygos.

Priešingai – saugumo savybės (angl. *safety*) yra pastoviai palaikomos. Saugumo savybė teigia, kad neįvyks nieko blogo jei yra išlaikoma esama būsena (t. y., kad per visus vykdymo etapus išlaikoma priimtina būklė). Kaip pavyzdys galėtų būti – užtikrinti, kad skrendančio lėktuvo aukštis visuomet būtų išlaikomas tam tikrose ribose (pvz. ne žemiau kaip 1 km. ir ne aukščiau kaip 4 km.).



Šaltinis: (Wooldridge, Jennings, Kinny, 2000)

3 pav. *Siūlomas daugiaagentinių sistemų kūrimo procesas pagal GAIA metodika*

Fig. 3*. The proposed multiagent systems development process based on GAIA methodology*

Tam, kad būtų įgyvendintos pareigos, vaidmuo turi eilę leidimų. Leidimai yra susiję vaidmeniui deklaruojamomis ir susijusiomis teisėmis. Vaidmenų leidimai identifikuoja turimus šaltinius, kad būtų įgyvendintos vaidmeniui priskirtos pareigos.

Tipiškai sumodeliuotose sistemose vaidmenims priskiriami leidimai dažniausiai yra gaunami iš informacinių šaltinių. Pavyzdžiui, vaidmuo gali būti susietas su sugebėjimu skaityti tam tikrą informacijos pavyzdį arba keisti kitą informacijos dalį. Vaidmuo taip pat gali turėti sugebėjimą generuoti informaciją.

GAIA metodikos tikslas - suteikti galimybę analitikui sistemingai pereiti nuo reikalavimų aprašymo iki sistemos specifikavimo, kuris yra pakankamai detalus, kad būtų galima įgyvendinti tiesiogiai. Taikant GAIA metodiką analitikas pereina visus sistemos kūrimo etapus nuo abstrakčių iki pakankamai konkrečių konceptų suformavimo ir sukūrimo žingsnių.

1.7.2. Intelektualiųjų įterptinių sistemų kūrimo metodika

Tai modifikuota GAIA metodika skirta intelektualiųjų įterptinių sistemų kūrimui. Šią daugiaagentinių sistemų kūrimo metodiką pristatė Laura Ostaševičiūtė disertaciniame darbe „Agentinėmis technologijomis pagrįstas intelektualiųjų įterptinių sistemų kūrimas“ (Ostaševičiūtė, 2009).

IĮS metodikoje, programinės ir techninės įrangos kūrimas yra traktuojami kaip atskiri procesai. Sąsajos su aparatūra užtikrinamos per specifinius bendruosius pakartotinai naudojamus komponentus - jutiklių ir valdiklių agentus bei techninės įrangos abstrakcijos lygmenį. IĮS metodika leidžia įvertinti techninei įrangai keliamus ribojimus. Metodika yra skirta konkrečiai intelektinių įterptinių sistemų klasei. Tai reiškia, jog metodika užtikrina specifinių dalykinės srities savybių modeliavimą ir specifikavimą. Lyginant su kitomis agentinėmis metodikomis, ši metodika leidžia įvertinti specifines intelektualiųjų įterptinių sistemų klasės ypatybes bei jų kūrimui keliamus reikalavimus.

Pagal šią metodiką yra sukurtas specifinis kūrimo karkasas JADE platformos aplinkoje. Programiniam realizavimui naudojama JAVA programavimo kalba.

1.7.3. Daugiaagentinių sistemų kūrimo metodika DIAMOND

Nagrinėjama daugiaagentinių sistemų kūrimo metodika DIAMOND (angl. *Decentralized Iterative Multiagent Open Networks Design)* yra skirta bendram

daugiaagentiniam aparatūrinės/programinės įrangos kūrimui, kadangi ji apjungia visos sistemos aparatūrinės ir programinės dalies projektavimą (Jamont, 2007).

Daugiaagentinių sistemos šiuo metu sparčiai plečiasi į belaidžių tinklų technologijų taikymo sritį. Todėl ir programinės, bendradarbiaujančios sistemos, tokios kaip išskirstyto valdymo arba apdorojimo, autonominės skaičiavimo ar interaktyvios sistemos šiuo metu vis dažniau grindžiamos belaidžio ryšio technologijomis.

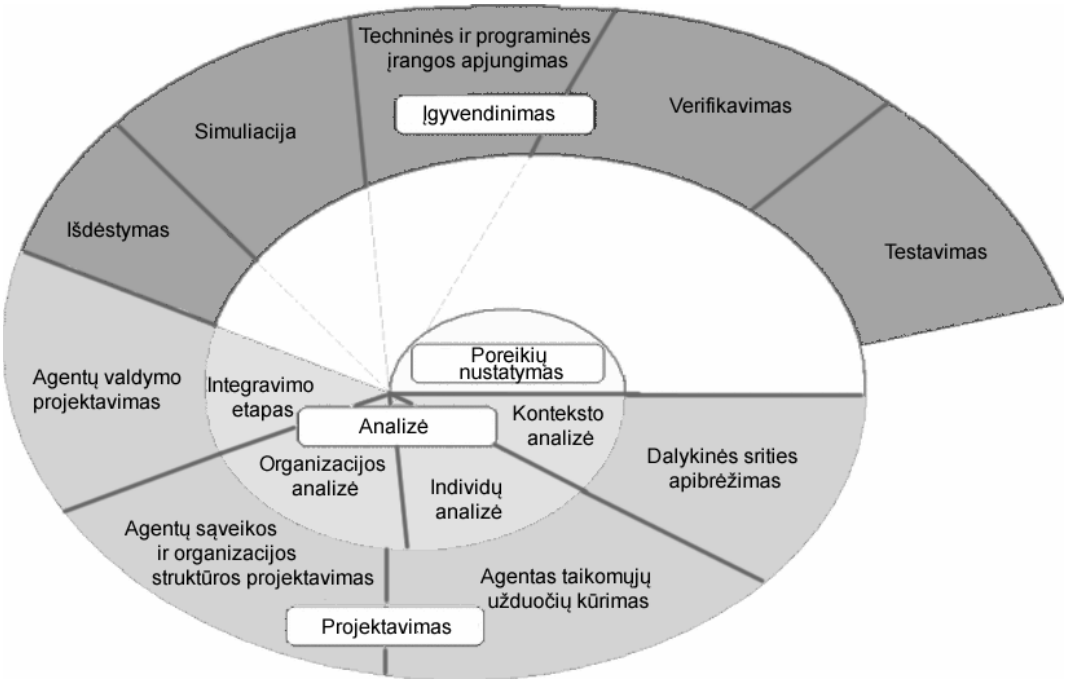
Jungiami į bendrą visumą techniniai įrenginiai yra vis labiau išskirstyti ir išdėstomi ne vienoje geografinėje vietoje. Tokių sistemų jungiančių atskirus įrenginius (sensorius, mikrovaldiklius, jutiklius ir pan.) į bendrą sistemą projektavimui ir realizavimui tenka pasitelkti decentralizuotų, išskirstyto intelekto sistemų kūrimo metodus ir metodikas.

Daugiaagentinės sistemos kūrimo ir gyvavimo ciklas pagal DIAMOND metodiką

Tradiciškai, kuriant įterptinę daugiaagentinę sistemą, sistemos programinė įranga yra kuriama naudojant daugiaagentinių sistemų kūrimo metodiką ir su jos vykdymu susijusius analizės, projektavimo testavimo ir kitus etapus. DIAMOND daugiaagentinės metodikos atveju į sistemą žiūrima kaip į visumą ir nesutelkiama dėmesio vien tik į programinės įrangos dalį.

Svarbiausia įterptinės sistemos kūrimo atveju yra sistemos komponenčių padalijimas į techninę ir programinę įrangą. Atlikdamas šią veiklą, projektuotojas turi nuspręsti, kurios sistemos dalys taps programine įranga, o kas – aparatine įranga. Tradiciniais metodais kuriant sistemą, padalijimas atliekamas kūrimo ciklo pradžioje. Tačiau, aparatinės įrangos ir programinės įrangos reikalavimas yra sudaromo remiantis visos sistemos reikalavimais. DIAMOND yra bendro aparatinės įrangos ir programinės įrangos kūrimo metodika, kadangi ji jungia aparatinės ir programinės įrangos kūrimą viso sistemos kūrimo ciklo metu. DIAMOND metodikoje šis padalijimo veiksmas atliekamas kūrimo ciklo pabaigoje.

Dėl sudėtingų sistemos savybių gyvavimo ciklo modelis turi leisti atlikti vėlyvus specifikacijų pakeitimus. Be to, būtina grįžti prie ankstesnių kūrimo etapų (tobulinimo) ir nagrinėti aparatinės įrangos / programinės įrangos apjungimo aspektus. DIAMOND kūrimo procese taiko daugybiškumą (t. y. prieauginį kriterijų). Skirtingų gyvavimo ciklo modelių įvertinimas šių ankstesnių kriterijų atžvilgiu nulemia spiralinio gyvavimo ciklo taikymą (Jamont, 2007).



Šaltinis: (Jamont, 2007)

4 pav. *Daugiaagentinės įterptinės sistemos gyvavimo ciklo modelis DIAMOND metodika*

Fig. 4. *Life cycle of multiagent embeded system development based on DIAMOND methodology*

DIAMOND metodika apima keturis pagrindinius etapus, paskirstytus spiralės formos gyvavimo cikle (4 pav.): sistemos reikalavimų nustatymą, daugiaagentinio pobūdžio analizę, bendrą aparatinės įrangos / programinės įrangos projektavimą su komponentais ir įgyvendinimas kurio metu atskiriama techninė ir programinė įranga (Jamont, 2007).

Sistemos reikalavimų apibrėžimo etapas

Sistemos reikalavimų apibrėžimo etapas DIAMOND metodikoje susideda iš dviejų žingsnių:

- fizinio lygmens sistemos turinio analizė;
- konkrečių sistemos režimų nustatymas.

Fizinio lygmens sistemos analizės metu nustatomi darbų srautų modeliai, pagrindinės užduotys, vartotojai ir kt. Tuomet nagrinėjami skirtingų aktorių veiklos tipai (vartotojų roles) ir bendrai naudojamų veiklų procesai bei su šiais procesais siejami informaciniai srautai. Šiame etape projektuojami vartotojų atvejų sąveikos modeliai, kurie gali būti kuriami taikant UML atvejų diagramas. Taip pat projektuojamos aktorių rolės, atliekamų paslaugų scenarijai,

pasitelkiant pagrindinius atliekamų rolių reikalavimus ir taikant UML sekų diagramų specifikavimo notaciją. UML sekų diagramos padeda aprašyti žinučių siuntimo scenarijus, siejamus su fizine informacinės sistemos struktūra ir išreiškia jų sąveiką.

Antrame žingsnyje atliekamas konkrečių sistemos režimų, kurie vadinami: veikimo režimais (angl. *running mode*) arba sustabdymo režimais (angl. *stop mode*) tyrimas. Paprastai pageidaujama, kad sistema veiktų autonomiškai. Jai veikiant su fizinėmis sistemomis reikia numatyti visą eilę konkrečių galimų elgsenos modelių. Kokia turi būti sistemos būsena atliekant techninę priežiūrą? Kaip kalibruoti sistemos komponentus? Kokia turi būti visų komponentų būsena sustabdžius sistemą avariniu būdu?

Sistemos analizės agentiniu aspektu etapas

Sistemos analizės agentiniu aspektu etapas vykdomas tuo pačiu metu dviejuose skirtinguose lygiuose. Visuomenės lygiu daugiaagentė sistema analizuojama kaip visuma, individualiu lygiu kuriami atskiri sistemos agentai. Šis integruotas daugiaagentinės sistemos kūrimo etapas DIAMOND metodikoje apima penkis pagrindinius toliau aptariamus žingsnius.

Situacijos analizės žingsnyje. apibrėžiama bendra situacija, t. y. aplinka, agentai, jų vaidmenys ir kontekstas. Pirmiausia nagrinėjamos aplinkos ribos, nustatomi pasyvūs ir aktyvūs komponentai. Šiame etape nustatoma, į kokius aplinkos elementus reikia atsižvelgti kuriamoje sistemoje (Russel, 1995; Wooldridge, 2000). Pirmiausia nustatomas aplinkos prieinamumo laipsnis, t. y. ką galima iš jos suprasti. Šios charakteristikų nurodo, kurie matavimai leidžia išmatuoti parametrus ir leis atpažinti aplinkos būseną. Šie elementai yra pagrindiniai nulemiantys agento sprendimo priėmimo aspektą. Aplinka gali būti apibrėžiama, jei ji yra agento nuspėjama, pradedant nuo esamos aplinkos būsenos ir nuo agento veiksmų kurie įtakoja aplinką. Fizinė aplinka retai būna deterministinė. Aplinka yra epizodinė, jei jos kita būsena nepriklauso nuo agento atliktų veiksmų. Ši savybė turi tiesioginę įtaką agento tikslams, kuriais siekiama valdyti aplinką. Reali aplinka beveik visuomet yra dinamiška, tačiau sistemos projektuotojas vienintelis gali įvertinti aplinkos dalies, kuria jis domisi, dinamiškumo lygį. Šis parametras turi didelę įtaką agento architektūrai. Fizinėms aplinkoms gali reikėti reaktyvių arba hibridinių architektūrų. Aplinka yra atskira, jei galimų aplinkos veiksmų ir būsenų skaičius yra baigtinis, tačiau reali aplinka dažniausiai yra tęstinė.

Sekančiame žingsnyje nustatomi aktyvūs ir pasyvūs sistemos elementai. Būtina nurodyti kiekvieno elemento vaidmenį sistemoje. Šiame žingsnyje nustatomi pagrindiniai

elementus, kurie bus naudojami sistemoje ir kurie tolimesniuose projektavimo etapuose taps agentais.

Individualaus agento projektavimo žingsnyje vykdomas sistemos procesų skaidymas nurodant agento išorinių ir vidinių aspektų skirtumą. Išoriniai aspektai yra susijęs su terpės, jungiančios agentą su išoriniu pasauliu, apibrėžimu (t. y. tuo, ką ir kaip agentas gali suvokti, ką jis gali perduoti). Agento vidinį aspektą sudaro apibrėžimas, kas agentui yra tinkama, t. y. ką jis gali padaryti (agento veiksmų sąrašas) ir ką jis žino (kitų agentų atstovavimas, aplinka, sąveika, organizavimas) (Demazeu 1995).

Daugeliu atvejų veiksmai yra atliekami remiantis turimais duomenimis apie agento aplinkos atstovavimą. Tokį atstovavimą, grindžiamą agento poreikiais, reikia nurodyti pateikiant veiksmų specifikacijas. Siekiant užtikrinti, kad duomenys būtų apdorojami realiu laiku, būtina apibrėžti šių veiksmų parametrus. DIAMOND metodika apibrėžia keturių tipų veiksmus.

Organizacijos analizės žingsnyje nurodoma sąveika tarp agentų. Sąveika vyksta perduodant pranešimus. Tokie mainų režimai yra formalizuojami. Protokolų aprašymai agentams yra išoriniai, tačiau jie dalijasi protokolų egzemplioriais. Toks bendravimas modeliuojamas naudojant AUML notacija.

Integravimo žingsnyje analizuojamas skirtingų agentų integravimas.

Projektavimo etapas

Šis etapas yra pagrįstas abstrakčiu komponentų suskaidymu. Abstraktų komponentą galime apibrėžti kaip elementarų objektą, kuris atlieka specifinę, tačiau pakartojamai panaudojamą funkciją. Jis sukurtas tokiu būdu, kad galėtumėte lengvai valdyti kitus komponentus ir sukurti programą. Komponentus galima sujungti vieną su kitu, kad būtų sudarytos sudėtingesnės funkcijos. Šiame žingsnyje pasirenkama skirtingų agentų architektūrą. Agentai kuriami vadovaujantis hibridine architektūra, t. y. sudėtine kai kurių atskirų tipų architektūrų struktūra.

Agentui taikomų užduočių kūrimo žingsnyje privaloma sukurti išorinį ir vidinį agento apvalkalą ir sudaryti bendravimo protokolus. Išoriniame agento apvalkale išsamiai aprašomos kiekvieno jutiklio ir vykdiklio sąsajos su išoriniu pasauliu. Šiame žingsnyje parenkamas jiems tinkamas technologinis sprendimas, nurodant visą informaciją apie gaunamą informaciją (iš jutiklių). Vidinio agento apvalkale aprašomi agento moduliai, prievadai jungiantys jį su išoriniu apvalkalu ar kitais komponentais ir moduliais.

Įgyvendinimo etapas

Įgyvendinimo etapą DIAMOND metodikoje sudaro keletą žingsnių. Pirmiausiai vykdomas abstrakčių komponentų, apibrėžtų trečiajame kūrimo etape, atskyrimas į programinę ir aparatūrinę įrangą. Bendro modeliavimo ir patvirtinimo žingsnyje vykdomas programinės įrangos, aparatinės įrangos ir jų sąsajų bendradarbiavimo modeliavimas. Šiame žingsnyje kiekvienas komponentas yra pilnai aprašomas, naudojant bendrąsias aparatinės įrangos ir programinės įrangos specifikacijas. Kiekvienam komponentui projektuotoja dažniausiai jau būna numatęs, kokiu būdu jie bus įgyvendinami: aparatine įranga ar programine įranga. Šiame žingsnyje būtina užtikrinti automatinį komponentų, kuriems pasirinktas programinės įrangos įgyvendinimas, kodo generavimą. DIAMOND metodikoje aparatinės įrangos komponentai aprašomi nudorojantis VHDL.

1.7.4. Kitos agentinių sistemų kūrimo metodikos

Be aukščiau aprašytų, GAIA (Zambonelli, 2003) ir DIAMOND, metodikų nagrinėjimui pasirinktos dar keturios daugiaagentinių sistemų kūrimo metodikos: PASSI (Cossentino, 2005), MaSE (DeLoach, 2004), Prometheus (Padgham, 2006) ir integruotos IĮS.

Šios metodikos buvo įvertintos pagal 2 lentelėje pateikiamus kriterijus kurie buvo aptarti 1.7 skyriuje.

1 lentelė. Daugiaagentinių sistemų kūrimo metodikų vertinimo kriterijai

X_1	Koks kūrimo metodikos gyvavimo ciklo modelis?
X_2	Kokius sistemos kūrimo etapus apima gyvavimo ciklas?
X_3	Kūrimo metodikos tipas („iš viršaus į apačią“ (angl. top-down), „iš apačios į viršų“ (angl. bottom-up))
X_4	Kokio tipo daugiaagentinėms sistemoms kurti skirta metodika?
X_5	Kokia didžiausia daugiaagentinė sistema gali būti sukurta naudojantis šia metodika?
X_6	Kokius agentų tipus gali projektuoti nagrinėjama metodika?
X_7	Ar yra sukurti daugiaagentinių sistemų modeliavimo įrankiai grindžiamų šia metodika?
X_8	Ar metodika leidžia specifikuoti kuriamos sistemos tikslus?
X_9	Ar metodika leidžia specifikuoti kuriamos sistemos roles?
X_{10}	Ar metodika laidžia specifikuoti skirtingus agentų tipus?
X_{11}	Ar metodika galima modeliuoti visą sistemą (agentus, išorinius aktorius, perceptus, scenarijus)?
X_{12}	Ar metodika leidžia specifikuoti agentų bendravimo protokolus?

X_{13}	Ar metodika leidžia specifikuoti agentų ir juos supančios aplinkos bendravimo ir duomenų paėmimo protokolus?
X_{14}	Ar metodika leidžia specifikuoti aparatūrinės įrangos keliamus ribojimus?
X_{15}	Ar metodika leidžia laiko reikalavimų specifikavimą?
X_{16}	Ar metodika leidžia įvykių prioritetų specifikavimą?

3 lentelė iliustruoja vertinimo rezultatus pagal kriterijus, susijusius su sistemų kūrimo procesu, visos septyniose analizuotose metodikos yra panašios. Jos visos naudoja interaktyvų procesų modelį ir yra pagrįstos koncepcijomis, išvestomis iš objektinio programavimo paradigmos. Visos metodikos apima detalaus projektavimo etapą, tik PASSI bei Prometheus apima ir realizacijos etapą. Visos metodikos, išskyrus IĮS ir DIAMOND, siūlo pagalbinius sistemų modeliavimo įrankius.

Vertinant metodikas pagal agentines savybes, šešios metodikos buvo palygintos pagal tikslus ir vaidmenis. Kaip nurodoma 3 lentelėje, tik MASE ir PROMETHEUS palaiko tikslų apibrėžimą, pagrįstą hierarchijos forma arba tekstiniais apibūdinimais. Tik IĮS ir DIAMOND metodikos turi galimybę nurodyti kaip vykdyti lygiagrečius tikslus ar kaip teikti jiems prioritetus. Kalbant apie vaidmenų identifikavimą, GAIA ir MASE siūlo stipriausią palaikymą įtraukiant vaidmenų modelius.

Iš pateikto metodikų vertinimo galima spręsti, jog tik IĮS (GAIA metodikos modifikacija) ir DIAMOND agentinės metodikos suteikia reikiamas priemones įterptinių realaus laiko sistemų kūrimui, tačiau nei IĮS nei DIAMOND neturi pagalbinių įrankių sistemos kūrimui.

2 lentelė. Agentinių sistemų kūrimo metodikų palyginimas pagal vertinamus kriterijus

	GAIA	**TROPOS**	**PROMETHEUS**	**PASSI**	**MASE**	**DIAMOND**	**IĮ**
Gyvavimo ciklas	Iteratyvus, etapai vykdomi nuosekliai	Iteratyvus, inkrementinis	Iteratyvus	Iteratyvus	Iteratyvus	Spiralinis	Iteratyvus visuose etapuose, etapai vykdomi nuosekliai
Kokius kūrimo etapus apima gyvavimo ciklas	Analizė, projektavimas	Analizė, projektavimas	Analizė, projektavimas, įgyvendinimas	Analizė, projektavimas, įgyvendinimas	Analizė, projektavimas	Analizė, projektavimas	Analizė, projektavimas
Kūrimo metodikos tipas	Iš viršaus į apačią	Iš viršaus į apačią	Iš apačios į viršų	Iš apačios į viršų	Iš viršaus į apačią	Iš viršaus į apačią	Iš viršaus į apačią
Kokio tipo sistemoms kurti skirta	Įvairioms	Įvairioms	Įvairioms	Įvairioms	Įvairioms	Įterptinėms sistemoms	Įterptinėms sistemoms
Kuriamos sistemos dydis	<= 100 skirtingų tipų agentų	neribojama	neribojama	neribojama	<= 10 skirtingų tipų agentų	neribojama	<= 100 skirtingų tipų agentų
Agentų tipai	Heterogeniniai	BDI tipo	BDI tipo	Heterogeniniai	Heterogeniniai	Heterogeniniai	Heterogeniniai
Modeliavimo įrankiai	Nėra (galima naudoti MASDK)	OpenOME	PDT	Nėra	agentTool	Nėra	Nėra
Tikslų specifikavimas	Nėra	Aktorių diagrama	Tikslų diagrama	Nėra	Tikslų hierarchinė diagrama	UML	Aprašas
Rolių specifikavimas	Rolių modelis	Nėra	Rolių diagrama	Agentų socialinis modelis	Rolių diagrama	UML	Vaidmenų specifikacija
Agentų tipų specifikavimas	Agentų modelis	Agento diagrama	Agento diagrama	Agentų socialinis modelis	Dokumentuotas aprašas	Aprašas	Vaidmenų specifikacija
Bendravimo protokolų specifikavimas	Bendravimo modelis	Sekų diagrama	Bendravimo protokolai	Agentų socialinis modelis	Bendravimo protokolai	Bendravimo protokolai	Sąveikos specifikacija
Sistemos architektūros specifikavimas	Nėra	Nėra	Sistemos diagrama	Įgyvendinimo modelis	Nėra	Aprašas	Sistemos išdėstymo diagrama
Agento-aplinkos bendravimo specifikavimas	Nėra	Nėra	Bendravimo protokolai	Nėra	Nėra	Aprašas	Nėra
Aparatūros keliamų ribojimų specifikavimas	Nėra	Nėra	Nėra	Nėra	Nėra	Nėra	Sąveikos specifikacija
Laiko reikalavimų specifikavimas	Nėra	Nėra	Nėra	Nėra	Nėra	Nėra	Aprašas
Įvykių prioritetų specifikavimas	Nėra	Nėra	Nėra	Nėra	Nėra	Aprašas	Aprašas

1.8. Pirmojo skyriaus išvados

1. Išnagrinėtos šešios agentinių sistemų kūrimo metodikos: GAIA (Zambonelli, 2003), PASSI (Cossentino, 2005), MaSE (DeLoach, 2004), Prometheus (Padgham, 2006), IĮS ir DIAMOND. Daugiaagentėje metodikoje paprastai naudojamos tik vienos kilmės, pavyzdžiui UML („Mase“, AAII, MESSAGE, PASSI), sąvokos ir modeliai. Kitose metodikose naudojama daug sąvokų, pavyzdžiui TROPOS („i*“ sąvoka gaunama inžinerinių žinių pagrindu, A-UML sąvoka skirta sąveikų protokolams ir planams) arba DESIRE (grafinė sąvoka, pagrįsta modeliavimo žiniomis, ir specifinė hierarchinė sąvoka užduočių aprašymams). Todėl norint apimti visas gyvavimo ciklo stadijas, skirtingiems abstraktumo lygiams reikia kelių formalizuotų kalbų.
2. Atlikta daugiaagentinių sistemų kūrimo metodikų analizė. Identifikuoti egzistuojančių agentinių metodikų trūkumai: metodikos yra per daug abstrakčios, nėra agentinės metodikos, atsižvelgiančios į svarbius šių sistemų ypatumus, tokius kaip sąsajos su aparatūra, jos keliami ribojimai ar laiko reikalavimų specifikavimas. Nuspręsta pasirinkti vieną tinkamiausią metodiką bei jos pagrindu sukurti specializuotą daugiaagentinės sistemos kūrimo metodiką.
3. Daugelis egzistuojančių daugiaagentinių metodikų paprastai išskiria tik analizės ir kūrimo stadijas. Labai mažai metodikų nagrinėja kitas stadijas. Pavyzdžiui, MASSIVE arba „Vowels“ galime rasti išdėstymo stadiją. Ši išdėstymo stadija įterptinių sistemų srityje yra labai svarbi, nes ji apima aparatinės / programinės įrangos padalijimą. Paskutinis ir didžiausias skirtumas tarp DIAMOND ir kitų daugiaagentinių metodų, yra tai, kad DIAMOND, priešingai nei kiti metodai, jungia aparatinės įrangos dalies ir programinės įrangos dalies vystymą. Kuriant tradicinę sistemą, padalijimas atliekamas ciklo pradžioje.
4. Siekiant apimti visą gyvavimo ciklą, skirtingiems dalykams skirtinguose lygiuose išreikšti reikalingi skirtingi formalumai. Dėl šios priežasties gyvavimo ciklas taikomas naudojant keturis etapus, kuriuose naudojant daugiau arba mažiau oficialias paradigmas ir kalbas, sumaišomos skirtingos išraiškos (agentai, komponentai, ribotos būsenos mašinos, aparatinės įrangos apibrėžimų kalbos). Naujausias esamas gyvavimo ciklas, naudojamas daugiaagentėse metodikose, yra klasikinis kaskadinis gyvavimo ciklas. Net jei kai kurie darbai bando pristatyti tokius pasikartojančius ciklus, kaip Kasiopėja (W) arba GAJA, DIAMOND spiralinio gyvavimo ciklo pasiūlymas yra labai originalus.

5. Apibrėžiant stadijos poreikius, į visuotinę veikimo sistemą įtraukiamas veikimo ir sustabdymo režimų tyrimas. Projektavimo etape leidžiama abstrakčiai elgtis su programine įranga ir aparatine įranga. DIAMOND metodika naudoja komponentus agentams kurti, kadangi mažai daugiaagenčių metodikų pateikia faktinį komponentinį matmenį. Šie komponentai naudojami kūrėjo darbui palengvinti naudojant vaizdinį programavimą, sudėtingumui valdyti atliekant funkcinį skaidymą, daugybiškumui padidinti pakartotiniu naudojimu, padalijimui palengvinti, nes lanksčiųjų komponentų ir lustų analogija leidžia aparatinės įrangos įrankiams ir programinės įrangos įrankiams vadovautis bendra vizija.
6. DIAMOND leidžia nagrinėti sritis, kuriose bendradarbiaujantys subjektai yra sujungti su fiziniais įrenginiais, kurie turi kontroliuoti arba prižiūrėti. DIAMOND naudoja UML naudojimo atvejus, nes juose pateikiamas patikimas poreikių apibrėžimas. Naudojimo atvejų schemų aiškinimai šiek tiek skiriasi nuo jų bendrojo naudojimo, nes aktoriai nėra įtraukiami į sistemą arba jos elementus. Be to, fizinių sąveikų atveju sąveikų schemoje negali būti aktorių.
7. Projektavimo stadijoje DIAMOND komponentus naudoja kaip operacinius vienetus. Šiuose komponentuose naudojamos ribotos būsenos mašinas arba komponentą, nustatytą apibūdinti vidinį veikimą. Šie formalumai leidžia generuoti programinės įrangos kodą arba aparatinės įrangos specifikacijas.

2. DAUGAAGENTINIŲ SISTEMŲ KŪRIMO PLATFORMOS

Antrajame skyriuje nagrinėjamos daugiaagentinių sistemų platformos, daugiau dėmesio skiriant jų darbinės aplinkos ir teikiamų funkcinių galimybių analizei, aprašomi svarbiausi šių sistemų komponentai. Atskleidžiami naudojamų daugiaagentinių sistemų kūrimo platformų privalumai bei trūkumai. Nagrinėjama, kaip taikyti daugiaagentinių sistemų kūrimo metodus nedidelio našumo įterptinių sistemų integravimui ir darbui belaidėje aplinkoje.

2.1. Daugiaagentinių sistemų kūrimo platformų samprata

Pasaulyje egzistuoja daug sukurtų agentinių sistemų, tačiau dauguma jų negali būti arba tik dalinai gali būti pritaikomos mažo našumo įterptinių sistemų kūrimui. Agentinė sistema FIPA-OS buvo sukurta realizuojant FIPA (O'Brien, 1998) standartus. Kita agentinė platforma APRIL Agent Platform (AAP), kitaip nei dauguma agentinių platformų, yra realizuota APRIL programavimo kalba (McCabe, 1995). AAP kūrimas ir plėtojimas yra nutrauktas, tačiau ši platforma išsiskiria dar vienu, prasmingu skirtumu nuo kitų sistemų: ji sudaro galimybes lengvai kurti ir diegti agentus internetinėje erdvėje bei palaiko internetinių paslaugų (angl. Web Services) ir semantinio žiniatinklio (angl. Semantic Web) standartus. Kita, vis dar kuriama daugiaagentinė platforma,- JASON (Bordini, 2005; Bordini, 2007). Jos pagrindinis privalumas yra lengvas BDI (angl. Belief-Desire-Intention) tipo agentų kūrimas (Rao, 1991). Platforma sukurta JAVA programavimo kalba, jos agentų elgsena aprašoma AgentSpeak programavimo kalba (Rao, 1996). JSON gali dirbti dviem būdais: pirmasis - kai visi agentai veikia viename kompiuteryje, antrasis būdas SACI (angl. Simple Agent Communication Infrastructure) pagalba leidžia išskirstyti agentus į skirtingas mašinas, SACI naudoja KQML kalbą (Finin, 1997). JADF (angl. JAVA Agent Development Framework) praktikoje dažniausiai taikoma agentinė platforma daugelio agentų sistemų kūrimui. JADE platforma pagrindinį dėmesį skiria FIPA modelio įgyvendinimui bei suteikia bendravimo infrastruktūrą ir papildomus įrankius: agentų valdymą, pagalbą jų kūrimui ir derinimui.

2.2. BDI tipo agentinės platformos

PRS seniausia BDI modeliu paremta daugiaagentinė platforma. Šia agentine platforma buvo realizuotos šios sistemos: PRS(Lisp realizacija), C-PRS(C realizacija), PRS-CL(Common Lisp realizacija), UM-PRS. PRS sistemos buvo pritaikytos kosminiuose keltuose

(space shutle). UM-PRS yra supaprastinta bei patobulinta PRS sistemos realizacija, parašyta C++ kalba. Ši platforma yra atviro kodo, bet nebeatnaujinama nuo 1996 metų. (Marcus, 2007).

dMARS platformą galima vadinti „antros kartos PRS sistema“. Platforma realizuota C++ programavimo kalba. Ši sistema buvo naudojama komerciniais tikslais, tačiau dėl neaiškių priežasčių nei pati sistema, nei jos išeities kodai nėra publikuojami internete (D'Inve M., 2004).

AgentSpeak(L) sukurta pagal PRS bei dMARS sistemas, pati programavimo kalba panaši į savo pirmtakes, tačiau turi daugiau konstrukcinių elementų, palengvinančių agentinį programavimą (Ancona, D., 2005).

JASON – patobulintas AgentSpeak(L) sistemos variantas. Tai atviro kodo sistema, gebanti interpretuoti AgentSpeak(L) kalba parašytą kodą, turinti ir daugiau papildomų galimybių. Sistema realizuota JAVA programavimo kalba.

SPARK (SRI Procedural Agent Realization Kit) – JAVA bei JYTHON paremta sistema. Panašiai kaip PRS agentinės programėlės, saugomos atskiruose aplankaluose su plėtiniu .spark. Jas interpretuoja SPARK interpretatorius. SPARK pagrindas yra PRS sistema.

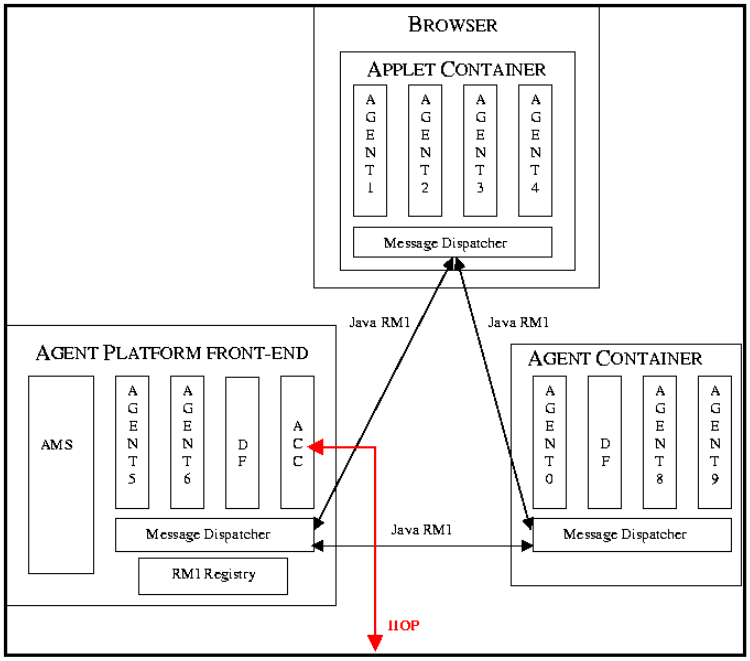
3APL sistema leidžia aprašyti sudėtingas sistemas, naudojant protingus agentus, viską aprašant intuityviais komponentais (įsitikinimais, tikslais ir planais). 3APL architektūra yra labai panaši į PRS sistemos, tačiau turi nemažai skirtumų, pavyzdžiui, PRS architektūra leidžia planuoti agentų tikslus (troškimus), o 3APL architektūra skirta valdyti agentų tikslų vykdymą. 3APL sistema yra realizuota JAVA bei HASKELL programavimo kalbomis.

2.3. JADE agentinė platforma

JADE (angl. *JAVA agent development framework)* - tai JAVA programavimo kalbos pagrindu sukurta daugiaagentinių sistemų kūrimo platforma, suderinama su FIPA standartais. JADE leidžia tinkle perduoti reikalingą informaciją ir resursus JAVA mobiliąsias aplikacijas palaikantiems įrenginiams: PDA, mobiliesiems ir išmaniesiems telefonams, nešiojamiesiems kompiuteriams ir panašiai. Agentų bendravimo aplinka JADE platformoje yra dinaminė, priklausomai nuo keliamų poreikių ir reikalavimų vykdymo metu gali kurtis ir išnykti agentai.

JADE struktūroje „run-time“ aplinkos reikalavimas vadinamas konteineriu, jis savyje saugo visus sukurtus agentus. Tokių konteinerių rinkinys vadinamas platforma (5 pav.). JADE lengvai suderinama su J2ME, J2EE ir CLDC. Agentai nereikalauja didelių atminties kiekių, todėl jie tinkami mobiliesiems prietaisams. Tam, kad platforma būtų suderinama su FIPA standartais, kiekvieną JADE konteinerį sudaro trys aptarnavimo agentai: sistemos valdymo agentas (AMS, angl. *Agent managment system*), tarpininko agentas (DFA, angl. *Directory*

Facilitator Agent) ir sistemos komunikavimą užtikrinantis agentas (ACL, angl. *Agent Communication Channel).* Dinamiškai naudojant katalogo tarpininko agentą, agentai gali aptikti kitus agentus ir bendrauti vienas su kitu, naudojant „peer to peer“ bendravimo būdą.



Šaltinis: (Bellifemine *et al.*, 1999)

5 pav. JADE agentinės platformos architektūra

Fig. 5. *Software architecture of JADE agent platform*

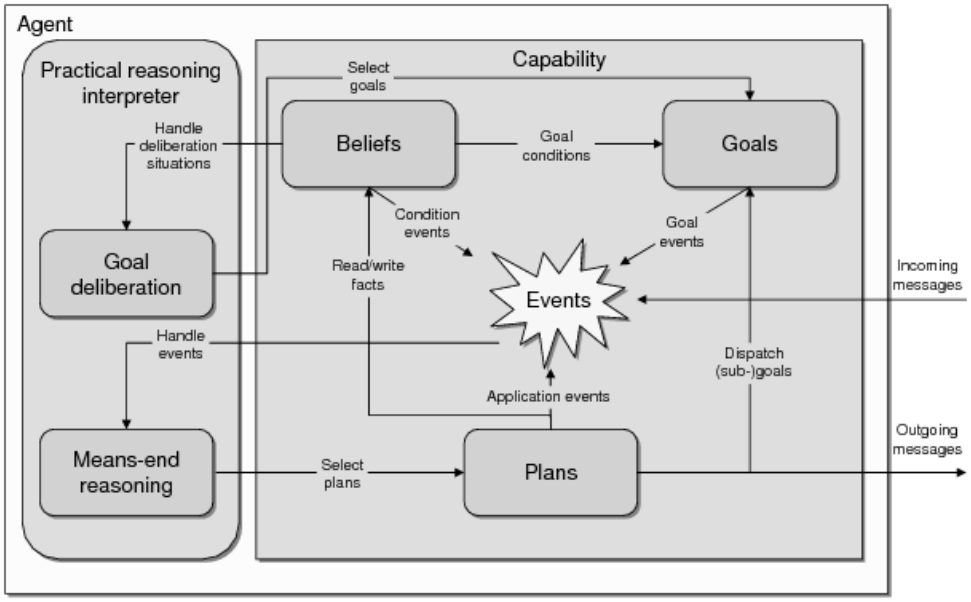
JADE agentai bendrauja naudodamiesi nesinchronizuotų žinučių perdavimo būdu. JADE palaiko stiprų autentifikavimo mechanizmą turinčią agentų apsaugą, kuri patikrina visų agentų teises. Žinutės, kuriomis agentai keičiasi informacija, turi būti parašytos agentų bendravimo kalba *(angl. Agent Communication Language*), atitinkančią FIPA standartus. JADE palaiko sudėtinių paralelių užduočių vykdymą toje pačioje JAVA gijoje, taip sumažindama techninės įrangos apkrovą. JADE palaiko agentų mobilumą, leisdama agentams persiųsti savo programinį kodą.

2.4. JADEX agentinė platforma

JADEX platforma skirta daugiaagentinių sistemų kūrimui, naudojant XML žymėjimo kalbą agentų aprašui ir JAVA programavimo kalbą jų veiklų aprašymui. Laikantis BDI modelio, JADEX protaujantys agentai yra atskiri programinės įrangos komponentai, turintys

aiškų tikslą. Pastarojo siekimui sukuriamos veiksmų sekos, iš kurių kiekviena veiksmų seka nurodo, kaip siekti užsibrėžtų tikslų esant įvairioms aplinkybėms. Agentas, siekdamas tikslo, pasirenka tą seką, kurią įgyvendinus bus gauta didžiausia nauda ir agentą labiausiai priartins prie siekiamo tikslo. Tačiau dažniausiai agentas nežino kiekvieno veiksmo būsimo rezultato. Pavyzdžiui, žaidime yra du pasirinkimo variantai, kur pirmasis saugus, mažą naudą turintis veiksmas, o antrasis mažiau saugus, bet didesnę naudą turintis veiksmas sėkmės atveju.

JADEX platforma palengvina daugiaagentinių BDI modelio sistemų kūrimą, leisdama tikslus ir planus aprašyti kaip bet kokios klasės objektus, kurie gali būti sukurti ir valdomi agento viduje. Agento planai realizuojami JAVA kalbos funkcijomis. JADEX agento architektūra pavaizduota 6 pav.



Šaltinis: (Braubach ir kt., 2005)

6 pav. *JADEX agento architektūra*

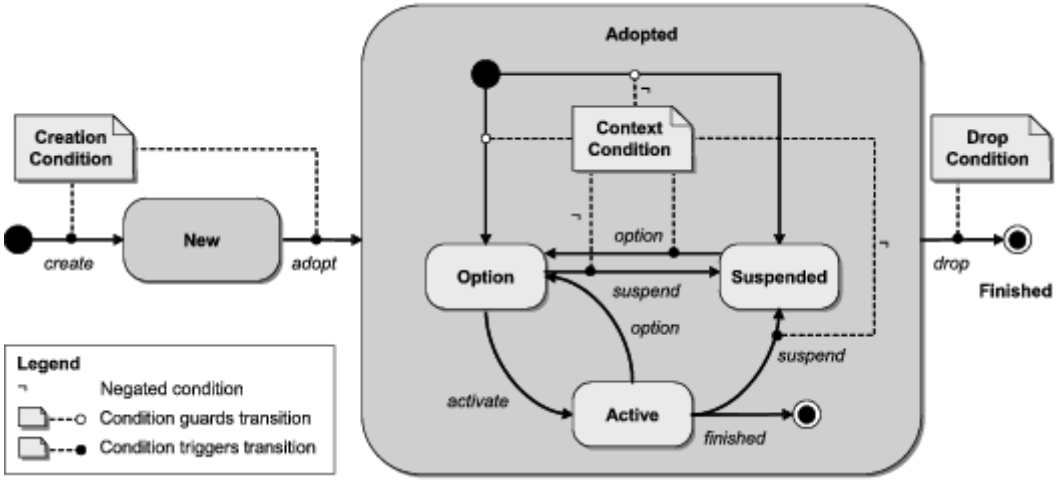
Fig. 5. *Architecture of JADEX agent*

Procesas, kurio metu pasirenkami JADEX platformoje agento įvykdomi planai, yra išskaldytas į dvi dalis: planų pasirinkimas, reaguojant į ateinančias žinutes, gautus duomenis, vidaus sistemos įvykius, ir planų pasirinkimas, siekiant įgyvendinti agento tikslus. Pagrindinės agento dalys JADEX platformoje yra įsitikinimai, tikslai ir planai.

Įsitikinimai - tai agento suvokimas apie jo pasaulį ir jį patį. Įsitikinimai sukuriami kaip JAVA kalbos objektai ir patalpinami į įsitikinimų duomenų bazę (ang. *belief base*). Pastaroji yra vieta, kurioje saugomi agento žinomi faktai. Įsitikinimai dažniausiai aprašomi ADF (angl.

Agent Definition File) faile XML žymėjimo kalba, o paleidžiami ir modifikuojami pasinaudojant agento planais.

Skirtingai nuo tradicinių BDI sistemų, kurios traktuoja tikslus kaip specialios rūšies įvykius, JADEX agentai laikosi pagrindinės idėjos, kad tikslai yra konkretūs laikini troškimai (Pokahr A. et al., 2005). Kai tikslas yra įgyvendinamas, jis tampa nuostata, išsaugoma agento įsitikinimų duomenų bazėje. Tikslo pasirinkimo mechanizmai yra atsakingi už visų agento tikslų valdymą, t. y., nusprendžia, kuris tikslas yra aktyvus, o kuris tik nuostata. JADEX sistema palaiko keturis pagrindinius tikslų tipus: atliekamas (angl. *perform*), pasiekiamas (angl. *achieve*), užklausiamas (angl. *query*), išlaikomas (angl. *maintain*) (6 pav.).



Šaltinis: (Pokahr, 2005)

7 pav. *JADEX agento tikslo gyvavimo ciklas ()*

Fig. 7. *JADEX goal life cycle*

Atliekami tikslai nurodo, jog agentas kažką turi vykdyti, tačiau tai neturi privesti prie kokio nors specialaus rezultato. Pavyzdžiui, šiukšlių rinkimo robotas gali turėti du bendrus tikslus vienu metu - judėti aplink ir ieškoti šiukšlių. Tokie tikslai būtų vykdomi nuolatos.

Pasiekiamas tikslas atskleidžia pasiekiamas galutines būsenas, tačiau jose nėra nurodoma, kaip šią būseną pasiekti. Agentas, siekdamas tikslo, gali bandyti skirtingas alternatyvas. Pavyzdžiui, agentas strateginiame žaidime, kuris turi tam tikrus ribotus išteklius, gali pasirinkti: derėtis su kitais žaidėjais arba bandyti pačiam rasti reikiamus išteklius.

Užklausiamas tikslas parodo, jog jo įgyvendinimui reikalinga tam tikra informacija. Agentui norint įgyvendinti tokį tikslą yra atrenkami ir vykdomi iš anksto aprašyti planai. Pavyzdžiui, robotui surinkus kelias šiukšles, reikia žinoti, kur yra artimiausia šiukšlių dėžė. Jei

robotas žino šią informaciją, jis juda tiesiai link dėžės, jei nežino, tuomet jis turi ieškoti, naudodamasis paieškos planu.

Išlaikomi tikslai nurodo būseną, kurią reikia išlaikyti, kai agentas ją pasiekia. Tai labiausiai abstraktus tikslas JADEX sistemoje. Pavyzdžiui, tjei agentas privalėtų neleisti reaktoriaus temperatūrai nukristi žemiau darbinės ribos ir pakilti aukščiau kritinės ribos, šis tikslas būtų aktyvuojamas, kai temperatūra pakilo aukščiau darbinės temperatūros. Agentas, vykdydamas įvairius planus, stengtųsi išlaikyti reaktoriaus temperatūrą tarp šių ribų.

JADEX agentų planuose aprašomi konkretūs agento veiksmai, kuriuos įgyvendinant galima pasiekti norimą tikslą. Planus sudaro dvi dalys: sąlygos, prie kurių planas gali būti vykdomas, ir pats planas, t.y., veiksmų seka, kurią agentas turi vykdyti. JADEX platformoje plano vykdymo sąlygos aprašomos XML kalba, o veiksmų seka JAVA funkcijomis (Braubach ir kt., 2005).

2.5. JACK agentinė platforma

JACK komercinė BDI modelio daugiaagentinių sistemų kūrimo platforma, sukurta naudojant JAVA programavimo kalbą. Naudojama BDI agentų architektūra gali būti lengvai praplečiama tam, kad palaikytų skirtingų agentų modelius ar specialius aplikacijų reikalavimus.

Pagrindiniai JACK platformos sukūrimo tikslai:

- Suteikti kūrėjams tvirtą, stabilų, lengvai naudojamą produktą;
- Patenkinti įvairius praktinių aplikacijų poreikius;
- Palengvinti technologijų perdavimą nuo trynėjimų prie industrijos;
- Įgyvendinti tolimesnius tyrimus.

Taip pat yra numatyta agentų savybė siekiant bendro tikslo jungtis į komandas. JACK aplinka naudoja specialiai sukurtą „BeliefSets“ struktūrizuotų užklausų duomenų bazę, kuri yra mobili bei gali judėti tinkle kartu su mobiliu agentu.

JACK agentai buvo suprojektuoti kaip komponentai, naudojami didesnėms sistemų aplinkoms. Todėl JACK sistemoje agentai egzistuoja kartu su kitomis programinėmis JAVA sistemomis ir yra matomi kaip paprasti JAVA objektai. JACK agentinė sistema leidžia lengvai naudotis visais sistemos resursais ir JAVA komponentais. Dėl panašių priežasčių JACK agentinė platforma nėra ribojama jokia specialiąja agentų bendravimo kalba.

2.6. PROMETHEUS kūrimo metodika ir daugiaagentinių sistemų platforma

PROMETHEUS metodikoje ir daugiaagentinių sistemų kūrimo platformoje yra numatyta palaikyti BDI modelio agentines konstrukcijas, tačiau pasinaudojant PROMETHEUS įrankiais galima kurti ir kitokių modelių agentines sistemas, išskyrus žemiausius konstrukcijos lygius, transformuojamus į programinį kodą. Kuriant kitų modelių agentines sistemas, žemiausias lygis turi būti modifikuotas rankiniu būdu. Pavyzdžiui, jei kuriamas JADE agentas, tai žemiausias lygis greičiau nurodys agento elgseną, o ne jo planus.

PROMETHEUS kūrimo metodiką sudaro trys fazės:

- Sistemos detalizavimas: sistema yra apibrėžiama naudojant tikslus ir galimų veiksmų planus. Specifikuojamos agentinės sistemos sąsaja su aplinka, veiksmais, aplinkos suvokimu ir išorine informacija. Yra apibrėžiamos funkcijos;
- Architektūrinis modelis: aprašomi agentų tipai ir visos sistemos struktūra, visa tai pavaizduojama bendroje sistemos diagramoje, sukuriami bendravimo protokolai;
- Detalus modelis: sukuriamos kiekvieno agento savybių detalės ir apibrėžiamos jų galimybės, duomenys, įvykiai ir planai.

PROMETHEUS metodikoje kuriami ne atskiri agentai, o jų tipai. Agento tipas yra formuojamas sujungiant keletą sistemos funkcijų į vieną grupę, kuri yra priskiriama tam tikro agento tipui. Tik galutiniame etape, t. y., realizacijoje, yra sukuriami atskiri agentai, remiantis aprašytų agentų tipais.

2.7. Agentinių sistemų modeliavimo kalbos

Agentinių sistemų analizei ir projektavimui dažniausiai naudojamos formalios specifikavimo kalbos: ConGolog ir CASL, Z kalba. Neformalūs agentinių sistemų kūrimo būdai dažniausiai remiasi struktūrizuota kalba ir grafinėmis notacijomis, pavyzdžiui, UML diagramomis. Yra sukurta keletas grafinių įrankių, specializuotų agentinių sistemų modeliavimui (PROMETHEUS, JAL ar AUML).

Agentinės sistemos gali būti realizuotos ir įdiegtos daugelyje platformų, tiek specializuotų standartais paremtų, tiek ir tradicinių - JAVA objektai arba komponentai. Realizacijai gali būti naudojamos įvairios programavimo kalbos. Tradicinės objektinės kalbos laikomos tinkamomis daugiaagentinių sistemų kūrimui, nes agento sąvoka yra artima objekto sąvokai, o specialios agentinės programavimo kalbos - FLUX, JACK ar 3 APL - laikomos

nauja kalbų klase. Agentinių sistemų realizacijai efektyviausia naudoti agentines platformas, kurios suteikia daugybę pakartotinai naudojamų komponentų ir servisų, reikalingų agentų realizacijai ir diegimui.

Pagrindinis skirtumas tarp agentų ir komponentų yra komunikavimo mechanizmas. Agentai naudoja komunikavimo kalbas, o komponentas - sąveikos protokolus. Agentinėje paradigmoje pranešimas siunčiamas siekiant perduoti dalį siuntėjo mentalinės būsenos gavėjui. Kai kitas agentas gauna pranešimą, jis gali tvirtinti, kad tinkamumo sąlygos tenkina siuntėjo lūkesčius ir kad siuntėjas stengiasi pasiekti atitinkamą racionalų efektą. Naudojant struktūrizuotas ACL, JADE platformoje, palengvinamas reaktyvių agentų, galinčių dalyvauti sudėtingose sąveikose, kūrimas.

Tiek agentams, tiek ir komponentams atsakomybės delegavimas priklauso nuo jų komunikacijos, o skirtingi delegavimo būdai pabrėžia jų komunikacijos modelių skirtumus. Komponentiniame modelyje siuntėjas pats atsakingas už pranešimo pasekmes. Kitaip tariant, komponentai nedeleguoja atsakomybės kitiems komponentams. Agentiniame modelyje pats gavėjas atsakingas už savo veiksmų pasekmes ir siuntėjas turi pasakyti, kodėl jis prašo serviso.

2.8. Daugaagentinių sistemų kūrimo platformų palyginimas

Daugiaagentinės platformos bus lyginamos pagal išsikeltus kriterijus:

- kokiai taikymo sričiai skirta agentinė platforma;
- kokių standartų buvo laikomasi, kuriant agentinę platformą. Kuo daugiau standartų platforma atitinka, tuo lengviau ji gali būti integruojama su kitomis programomis ar sistemomis;
- kokioje operacinėje sistemoje agentinė platforma gali veikti. Šis kriterijus apibrėžia agentinių platformų suderinamumą su mobiliaisiais įrenginiais (jei ji įgyvendinta JVM);
- programavimo kalbos, kuriomis realizuota agentinė platforma ir kurios naudojamos daugiaagentinių sistemų kūrimui. Nedidelio našumo įterptinėse sistemose negali būti vykdomos aukšto lygio programavimo kalbos;
- kokie bendravimo protokolai yra įgyvendinti daugiaagentinėje platformoje. Kuo daugiau bendravimo protokolų palaiko agentinė sistema, tuo lengviau sukurti komunikaciją tarp agentinės sistemos ir kitos programinės įrangos;
- kokio tipo licencija taikoma platformai;
- ar platforma platinama atviru kodu;

– kada paskutinį kartą buvo atnaujinta daugiaagentinių sistemų kūrimo platforma.

Agentinių platformų palyginimo rezultatai pateikti 3 lentelėje.

3 lentelė. Daugiaagentinių sistemų kūrimo platformų palyginimas

	Gamintojas	Taikymo sritis	Standartai	Operacinė sistema	Programavimo ir specifikavimo kalbos	Bendravimo protokolai	Licenzijos tipas	Atviro kodo	Paskutinis leidimas
AgentScape	Delft Technologijos universitetas	Didelės apimties išskirstytos agentės sistemos	nėra	JVM	Java, XML	Message exchange (platform defined syntax)	BSD	Taip	2013.07.05
Swarm	Swarm Development Group	Bendrosios paskirties agentinės sistemos	nėra	Windows, Linux	Java	Message exchange (platform defined syntax)	GPL	TAIP	2005.02.12
JACK	AOS	Dinamiškos ir sudėtingos aplinkos	FIPA	Windows, Unix, JVM	Java, JAL, XML	DCI tinklas, TCP/IP	Komercinė	NE	2010.09.02
JADE	Telecom Italia (TILAB)	Išskirstytos sistemos sudarytos iš autonominių agentų	FIPA, CORBA	JVM	Java	ACL, MTPs,RMI, IIOP, HTTP, WAP	LGPLv2	TAIP	2013.12.06
Jadex	Hamburgo universitetas University	Išskirstytos sistemos sudarytos iš BDI tipo agentų	FIPA, SOA, WSDL	JVM	Java, XML	HTTP	LGPLv2	TAIP	2013.12.20
MaDKit	Institut universitaire de technologie	Agentų imitaciniam modeliam	UML	JVM	Java, C/C++, Python	peer-to-peer	GPL	Taip	2014.04.21
JIAC	Berlyno technikos universitetas	Didelės apimties išskirstytos agentinės sistemos	Nėra	JVM	Java, XML	ActiveMQ	Apache License V2	Taip	2014.12.12
Jason	Santa Catarina universitetas	Išskirstytos sistemos sudarytos iš BDI tipo agentų	Dalinai FIPA	Windows, MacOS, Linux	Java, AgentSpeak	speech-act based, KQML	LGPLv2	Taip	2013.12.12

2.9. Antrojo skyriaus išvados

1. Dauguma siūlomų agentinių sistemų kūrimo platformų yra skirtos didelį pajėgumą turinčių įterptinių sistemų, veikiančių esant stacionarioms sąlygoms, turinčių dideles duomenų saugyklas (kompiuterinės atminties), kūrimui. Jų realizacijai dažniausiai taikomos aukšto lygio programavimo kalbos (pvz. JAVA), tačiau jos nėra tinkamos ribotų galimybių įterptinių sistemų darbo užtikrinimui.
2. Skyriuje nagrinėjamos agentinių sistemų kūrimo platformų galimybės. Įvertinus platformų savybes, buvo nustatyta pradinio kodo generavimui tinkamiausia PDT aplinka (PROMETEUS metodika grindžiamas įrankis, skirtas agentų modeliavimui ir programinio kodo generavimui), įdiegiama į ECLIPSE IDE. Programinis kodas, atsižvelgiant į nutylėjimą, generuojamas JAVA programavimo kalba, tačiau yra galimybė generuoti ir C++ programavimo kalba, grindžiama programiniu tekstu, tuomet palengvėja agentinės sistemos pernešimas į nedidelio našumo įterptinę sistemą

3. TINKAMOS REALAUS LAIKO OPERACINĖS SISTEMOS PARINKIMAS NEDIDELIO NAŠUMO ĮTERPTINIŲ SISTEMŲ INTEGRAVIMUI

Skyriuje nagrinėjamos skirtingos realaus laiko operacinės sistemos ir jų tinkamumas, siekiant užtikrinti nedidelio našumo įterptinių sistemų darbą. Remiantis atlikta OS savybių analize ir jų našumo eksperimentiniais tyrimais nustatyta, jog tinkamiausia DAS kūrimui yra operacinė sistema FreeRTOS.

3.1. Realaus laiko operacinių sistemų samprata

Realaus laiko operacinė sistema, RTOS – operacinės sistemos, garantuojanti programoms minimalų uždelsimą, kuris gali trukti tarp momento, kai tam tikras įrenginys sugeneruoja užklausą iki tol, kol programa pradės apdoroti šią užklausą. Realaus laiko operacinė sistema veikia kaip ir bet kuri kita operacinė sistema. Tai programinė įranga, turinti taikomųjų programų sudarymo sąsajų (API) rinkinį, kurį programuotojai gali naudoti kurdami sistemas bei sprendimus. Šios operacinės sistemos ypatingos tuo, kad padeda įterptinėms sistemoms atitikti galutinius terminus. RTOS pasirinkimas konkrečiam atvejui yra atliekamas analizuojant tam tikrus jų parametrus, lemiančius jos veikimą. Tam tikroms sistemoms gali būti reikalingas mažiausias pertraukties gaišties laikas, tačiau tai suteikianti RTOS gali turėti didesnę prioritetinės inversijos tikimybę. Galima išskirti šiuos pagrindinius reikalavimus kuriuos privalo tenkinti realaus laiko operacinės sistemos:

- turi galėti vykdyti keletą užduočių vienu metu, kurių veikimas turi būti pilnai kontroliuojamas OS;
- turi valdyti procesų prioritetus;
- žinomas maksimalus uždelsimo laikas, tarp įrenginio sugeneruoto pertraukimo ir valdymo perdavimo tvarkyklei;
- žinomas maksimalus laikai, kuriuos tvarkyklė sunaudos pertraukimui apdoroti;
- žinomas bendras pertraukimo uždelsimas (laikas praėjęs nuo pertraukimo pradžios iki vykdymo perdavimo programai).

RTOS yra trijų tipų: realaus laiko kieta (angl. *Hard real-time*), realaus laiko tvirta (ang. *Firm real-time*) ir realaus laiko minkšta (angl. *Soft real-time*). Kietoje realaus laiko

sistemoje užduotys ir pertraukimo užklausos turi būti atlikti iki nurodytų terminų. Vieno termino nesilaikymas gali lemti visos sistemos gedimą. Tvirta realaus laiko sistema leidžia nesilaikyti keleto terminų, tačiau praleidus daugiau terminų, gali įvykti sistemos lūžis. Minkšta realaus laiko sistema yra tokia, kurioje dažniausiai laikomasi terminų, tačiau šis apribojimas nėra labai griežtas (Almeida, 2004).

Įterptinių sistemų programinė įranga dažniausiai kuriama dviem būdais: pagrindinio ciklo su pertrauktimis arba panaudojant realaus laiko operacines sistemas. Pagrindinio ciklo su pertrauktimis metode naudojamas vienas pagrindinis, nuolat vykdomas ciklas, o užduočių paralelizmas pasiekiamas pertraukimų, kuriuos apdorojus grįžtama į pagrindinį ciklą, pagalba. Tokios realizacijos pagrindiniai trūkumai:

- Programą sunku išskaidyti į atskirus modulius;
- Sudėtingesnis programavimas ir programos tobulinimas;
- Naudojami globalūs kintamieji.

Tokios realizacijos privalumai:

- Gali būti parinktas optimalus realaus laiko įvykių aptarnavimo scenarijus, todėl reakcijų į įvykius trukmės gali būti minimalios;
- Mažesni reikalavimai sistemos resursams (procesoriaus greitaveikai, atminties dydžiui).

Realizuojant sistemą panaudojus realaus laiko operacines sistemas, atsiranda galimybė dirbti su patogiais ir lengvai pritaikomais programavimo šablonais, skirtais taikomosios programinės įrangos kūrimui. RTOS pagalba valdomas užduočių skirstymas bei abstrahuojamas aparatūrinės įrangos valdymas (taikomųjų programinių sprendimų izoliacija nuo aparatūrinės įrangos detalių) (Sha, 2004).

3.2. Realaus laiko operacinių sistemų savybės ir funkcinės galimybės

Kiekviena RTOS turi branduolį (ang. *Kernel*), kuris tarnauja kaip jos šerdis (ang. *Core*). Šerdis yra apsupta apvalko sluoksnių, kurie suteikia apsaugą ir leidimus prie prieigos. RTOS gamintojai suteikia rinkinį taikomųjų programų programavimo sąsajų (angl. *Application Programming Interface* (API)), kurios naudojamos norint pasiekti šį branduolį ir atlikti užduotis. RTOS atmintis yra padalinta į branduolio erdvę, kurią sudaro branduolio kodas, ir vartotojo erdvę, kurią sudaro vartotojo kodas. Gijos realaus laiko operacinėje sistemoje yra kaip funkcijos, turinčios savo laikinąją informacijos saugyklą ir gijos valdymo bloką (ang. *Thread Control Block*). Branduolį sudaro planuoklis, kontroliuojantis šias gijas nuosekliu

būdu. Kiekviena RTOS turi pagalbinių įrankių, skirtų daugiaprogramiams režimui (gijoms), jie dažniausiai palaiko gijos sinchronizavimą, naudojant semaforus arba muteksus (angl. *mutex*). Kiekviena RTOS privalo turėti pakankamą skaičių prioritetinių lygių ir vengti prioritetinės inversijos.

Įvykiu gali būti laikoma aparatinės įrangos pateikta pertraukimo užklausa arba operacinės sistemos atliekama užduotis. Kai kalbama apie aparatinės įrangos pertraukimą, gaišties laiku (ang. *Event Latency*) vadinamas intervalo tarpas nuo pertraukimo užklausos pateikimo iki pertraukimo apdorojimo procedūros kodo pirmosios eilutės vykdymo. Kalbant apie sistemos įvykį, gaišties laikas yra laiko tarpas nuo signalinės užduoties sugeneravimo iki užduoties pirmojo nurodymo įvykdymo. Vėlinimo svyravimas (ang. *Jitter*) yra bet koks fiksuoto laiko tarpo, skirto užduoties įvykdymui, pertraukimo apdorojimui ir t. t., nuokrypis.

Pirmenybinė inversija yra didelė daugumos RTOS problema, atsirandanti tuomet, kai mažesnio prioriteto užduotis užkerta kelią didesnio prioriteto užduočiai ir pakeičia prioritetinę struktūrą. Dauguma RTOS suteikia ypatingus metodus, siekiant išvengti pirmenybinės inversijos. Dažniausiai pasitaikantys metodai yra pirmenybės lubos *(*angl. *priority ceiling)* ir pirmenybės paveldėjimas *(*angl. *priority inheritance)* (Davis, 2005). Pirmenybės lubose kiekvienam ištekliu besidalinančiam procesui yra paskiriamas prioritetas, lygus didžiausio prioriteto procesui, kuris gali užrakinti išteklių. Pirmenybės paveldėjimo atveju, jei didesnio prioriteto užduotis laukia vykdomos mažesnio prioriteto užduoties, mažesnio prioriteto užduočiai laikinai (iki jos įvykdymo) paskiriama laukiančios didesnio prioriteto užduoties pirmenybė, kad vidutinio prioriteto užduotis neužkirstų jai kelio ir nepaveiktų didesnio prioriteto užduoties.

RTOS užduotims, kurioms reikalinga atmintis, paskiriama iš sistemoje esančios atminties. Atlikus atminties reikalaujančią užduotį bei po įvykdymo, jos daugiau nebereikia, atmintis turi būti perskirstyta ir vėl prieinama kitoms gijoms. Siekdami užtikrinti šį procesą, skirtingi branduoliai naudoja skirtingus algoritmus. Du dažniausiai pasitaikantys modeliai RTOS sistemose yra plokščios atminties modelis *(*ang. *Flat Memory Model)* ir segmentinio adreso modelis *(*ang. *Segmented Address Model).*

Tam tikros kritinės kodo dalys gali bandyti naudoti vieną bendrą išteklių. Betarpiškai atskiriančios prieigą prie kompiuterinių resursų žymos apsaugo bendrą išteklių užtikrindamos, kad dvi užduotys tuo pačiu metu negalėtų pasinaudoti šiuo ištekliumi. Semaforas yra skaitinių žymų rinkinys, galintis reguliuoti laukiančius eilėje prie kompiuterinio resurso procesų. Semaforai valdo prieigą prie bendro ištekliaus, į eilę suskirstydami užduotis, kurioms reikalinga ši prieiga.

3.3. RTOS tinkamų nedidelio našumo įterptinėms sistemoms analizė

Tolimesnei analizei buvo pasirinktos šešios realaus laiko operacinės sistemos: Nano-RK, TinyOS, FreeRTOS, eCos, FreeRTOS, Contiki, Mantis (4 lentelė).

3 lentelė. RTOS, skirtos mažo našumo įterptinėms sistemoms

Operacinė sistema	Kūrėjas	Laisvai platinama	Palaikoma	Naujausia versija, Išleidimo data
TinyOS	Berklio Universitetas	+	+	2.1.2. 2012.08.08
Contiki	Švedijos kompiuterijos institutas	+	+	2.7 2013.11.15
Mantis	Kolorado universitetas	+	-	1.0b 2012.07.03
Nano-RK	Carnegie Mellon universitetas	+	+	Rev 877 2009.07.15
eCOS	eCosCentric	+	+	3.0 2009.03.30
FreeRTOS	John Westmoreland	+	+	8.2.1 2015.03.24

3.3.1. Realaus laiko sistema Nano-RK

Nano-RK yra sukurta siekiant suteikti kietą realaus laiko veikimą LINUX pagrindu. Linux branduolys yra monolitinis su moduliais. Visa operacinė sistema yra padalinta į dvi sudedamąsias dalis. Realaus laiko dalis veikia realaus laiko branduolyje, o nerealaus laiko dalis veikia Linux sistemoje. Šios dvi sudedamosios dalys komunikuoja „pirmas įėjo, pirmas išėjo" (ang. first in, first out) (toliau FIFO) buferiais, vadinamais realaus laiko FIFO buferiais. Realaus laiko branduolys yra tarp aparatinės įrangos ir Linux branduolio ir taip pasirūpina visais pertraukimais. Jei pertraukimas iškviečia realaus laiko užduotį, tai realaus laiko branduolys užkerta kelią Linux sistemai (jei programa tuo metu veikia) ir suteikia galimybę įvykdyti realaus laiko užduotį (Eswaran, 2005).

RTLinux yra suderinamas su ribotu skaičiumi architektūrų, pavyzdžiui, x86, PowerPC, MIPS, Alpha, skirtingai nuo eCos. Linux sistema apskritai gali būti suderinama tik su 100 pirmenybės lygių. Planuoklis naudodamas primityvus „sched_get_priority_min()“ ir „sched_get_priority_max()“, gali nustatyti didžiausius ir mažiausius prioritetus. Naudodamas realaus laiko plėtinį, branduolys gali būti suderinamas su prioritetiniu FIFO planuokliu ir

praplėsti iki 1024 pirmenybės lygių. RTLinux turi kieto realaus laiko pertraukimo gaišties laiką. Visi kiti gaišties laikai priklauso nuo centrinio procesoriaus.

Įprastinis Linux branduolys nėra suderinamas su pirmenybėmis pagal realaus laiko apribojimą, išskyrus tuos atvejus, kai vykdomas tik vartotojo kodas. Norint įgyvendinti visišką prevenciją, reikia įdiegti „CONFIG_PREEMPT“ programos ištaisymui skirtą kodą. Tai leidžia naudotojo kodui ir branduolio vietos kodui anksčiau už kitus pasinaudoti didesnio prioriteto realaus laiko gijomis, todėl blogiausio atvejo gaišties laikas sumažėja iki maždaug kelių milisekundžių. Tačiau šio metodo trūkumas toks, kad branduolio kodas yra aukštesnio prioriteto, todėl atsiranda daug besikaitaliojančio konteksto, sumažinančio visos sistemos našumą.

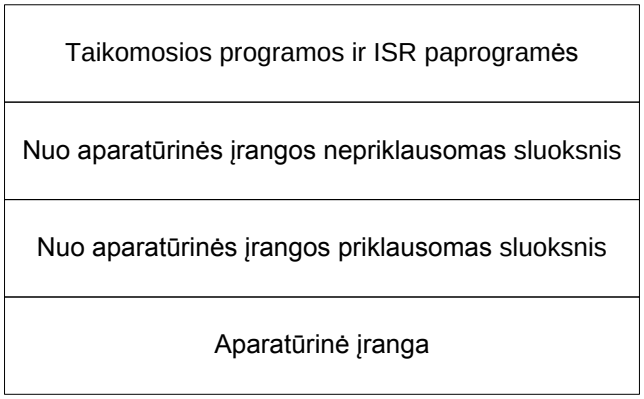
Nano-RK sistemoje pertraukimo mechanizmas yra suskirstytas į dvi dalis: apatinė dalis priima pertraukimo užklausą iš išorinių įrenginių ir išsaugo užklausą atminties buferyje, viršutinė dalis perskaito šį buferį ir perduoda informaciją į branduoliui prieinamą buferį. Visos pertraukimo įgalinimo/apribojimo užklausos, kurias pateikia Linux branduolys, nėra perduodamos techninei įrangai, bet jos yra emuliuojamos. Pavyzdžiui, jei Linux apriboja aparatinės įrangos pertraukimą, jis iš tikrųjų liks įgalintas ir realaus laiko branduolys patalpins šį užklausimą į eilę, todėl pristatymas bus atidėtas. Tačiau realaus laiko pertraukimai nėra paveikiami, o juos įprastu būdu prižiūri realaus laiko branduolys. Visos RT Linux OS turi modifikacijas, kurios sumažina pertraukimo gaišties laiką ir vėlinimo svyravimą.

Gijos sinchronizavimas ir prioritetinė inversija. Nano-RK gijų sistema sinchronizavimui naudoja muteksus, sujungimus ir būklės kintamuosius. Sujungimų pagalba gijos priverčiamos laukti tol, kol kita gija bus pabaigta. Būklės kintamųjų mechanizme tam tikras kintamasis nustatomas kaip tiesa ir gijos bus vykdomos iki tol, kol ši reikšmė išliks nepakitusi. Būklės kintamasis visada naudojamas kartu su muteksu, siekiant išvengti aklaviečių ir lenktyniavimo. Norint apeiti pirmenybės inversiją Nano-RK sistemoje, naudojamos abi: pirmenybės paveldėjimo ir pirmenybės lubų technikos.

3.3.2. Realaus laiko operacinė sistema FREE RTOS

„FreeRTOS“ sistema suderinama su daugybe sistemų: Intel, ARM, Atmel, PIC, Xilinx ir t. t. Ji suderinama su pirmumo bei bendradarbiavimo planavimu ir, siekiant atitikti reikalavimus, gali būti konfigūruojama. FreeRTOS sistemą sudaro trys sluoksniai (6 pav.): nuo aparatūrinės įrangos priklausomas sluoksnis (keičiamas keičiant techninę įrangą (mikrovaldiklį), jo pagalba bendraujama su technine įranga), nuo aparatūrinės įrangos

nepriklausomas sluoksnis (šiame sluoksnyje yra realizuotas procesų planuoklis, failinės sistemos palaikymas, API ir kita programinė įranga), taikomųjų programų ir ISR paprogramių sluoksnis (šiame sluoksnyje veikia vartotojo programinė įranga bei pertraukimų apdorojimo paprogramės (ISR), šis sluoksnis bendrauja su nuo aparatūrinės įrangos nepriklausomu sluoksniu) (Mohamadi, 2011).



Šaltinis: sudaryta pagal (Mohamadi, 2011)

8 pav. FreeRTOS programinės įrangos sluoksniai

***Fig. 8.** FreeRTOS software layers*

Planuoklis „FreeRTOS“ sistemoje gali veikti dvejais režimais: išstumiančiu (*angl. preemptive*) ir bendradarbiavimo (*angl. cooperative*), režimas pasirenkamas konfigūracijos pagalba (Inam, R. 2011). Naudojant bendradarbiavimo planavimą, išvengiamos pakartotinio įėjimo problemos, su kuriomis susiduriama išstumiamame planavime. Tai vyksta dėl to, kad vykdomos užduotys gali būti pertraukiamos tik tose vietose, kurias leidžia programuotojas. Svarbu suprasti, jog šiuo būdu realaus laiko veikimas yra suvaržytas užduočių lygmenyje, tačiau pertraukimai, naudodami semaforus, toliau gauna realaus laiko atsakus. Iš pradžių atliekamos didžiausio prioriteto užduotys, daugiau nei vienai didelio prioriteto užduočiai atlikti naudojamas žiedo (*angl. round robin*) mechanizmas.

Sukūrus bet kokią užduotį „FreeRTOS“ sistemoje, branduolys atlieka du atminties paskirstymus. Laikas, kurio reikia atminties paskyrimui užduoties valdymo bloke, yra fiksuotas. Laikas, kurio reikia užduoties atlikimui, yra proporcingas užduoties sudėtingumui, t. y., reikalingas užduočių kiekis. Be to, pirmosios užduoties sukūrimo gaišties laikas yra didžiausias, vėliau jis sumažėja ir yra lygus gaišties laikui, kurio reikia vienodos apimties užduotims.

„FreeRTOS“ suderinama su trimis atminties vietos paskyrimo modeliais. Paprasčiausi modeliai paskiria fiksuotą atmintį kiekvienai užduočiai, tačiau jos neperskirsto užduočiai pasibaigus. Tokiu būdu atmintis negali būti naudojama pakartotinai, o tai lemia atminties sumažėjimą. Antrasis modelis leidžia paskirti ir perskirstyti atmintį bei naudoja geriausios atitikties algoritmą, siekiant surasti vietą atmintyje. Sudėtingiausias modelis naudoja nestandartinius, kiekvienai užduočiai sukuriamus algoritmus, siekiant atitikti specifinius reikalavimus.

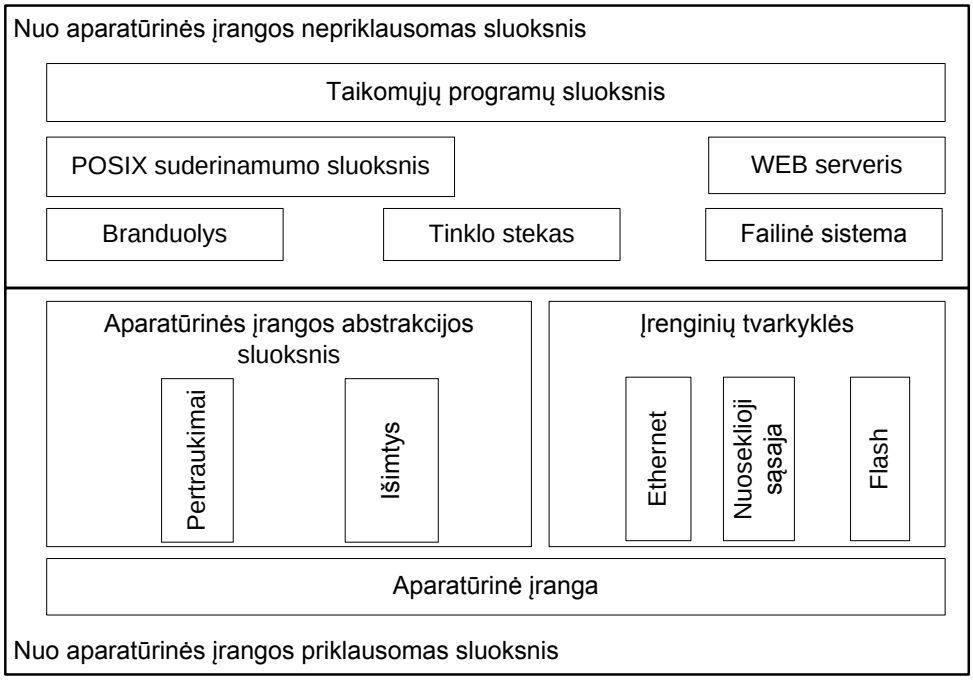
Manoma, jog didžiausias „FreeRTOS“ sistemos trūkumas yra tai, kad ji neįdiegia pažangių metodų, pavyzdžiui, pirmenybės lubų, siekiant išvengti pirmenybės inversijos. Todėl šią sistemą gali būti rizikinga naudoti kritinėse situacijose.

3.3.3. Realaus laiko operacinė sistema eCos

„eCos“ realaus laiko operacinės sistemos struktūra yra pagrįsta moduliniu ir sluoksniniu realaus laiko branduoliu. Jos pagrindinis išskirtinumas yra konfigūravimo sistema, leidžianti įkompiliuoti į sistemos branduolį tik tuos komponentus, kurių reikia specifinėms programoms (Wu Chen 2011). Siekiant šių didelių konfigūravimo galimybių užtikrinimo, prijungtas specialus aparatinės įrangos abstrakcijos sluoksnis. Tai garantuoja atskyrimą tarp struktūros ir platformos bei suteikia nepriklausomą prieigą prie jų abiejų, taip likusi sistemos dalis lieka visiškai nepriklausoma nuo įrengimo. Be to, tai leidžia nesudėtingai perkelti sistemą į įvairias technines platformas. Norint pasirinkti komponentus, kurie bus montuojami į branduolį, „eCos“ suteikia galimybę naudotis grafine vartotojo sąsaja, vadinama „eCos“ konfigūravimo įrankiu, taip supaprastinant konfigūravimo užduotį ir gaunant puikią branduolio adaptaciją. Jos taikomųjų programų programavimo sąsajos yra suderinamos su operacinės sistemos sąsajos POSIX standartu. eCos realaus laiko operacinės sistemos struktūra pateikiama 9 paveiksle.

„eCos“ gali būti sudaryta iš dviejų planuoklių: taškinio atvaizdo ir daugiapakopės eilės planuoklio. Jie abu yra suderinami su išstumiančiu (*angl. preemption*) režimu, tačiau į sistemą vienu metu įdiegiamas tik vienas planuoklis. Siekiant apsaugoti planuoklio prieigą, pertraukimai gali būti apribojami kritinėse srityse. Deja, tai padidina didžiausią išsiuntimo gaišties laiką. Siekiant to išvengti, naudojamas skaitiklį turintis mechanizmas. Taškinio vaizdo planuoklį sudaro 32 lygių pirmenybės eilė, kai 0 reiškia didžiausią pirmenybę, o 31 – mažiausią. Kiekvienai gijai priskirta tam tikra pirmenybė, todėl vienu metu gali būti vykdomos tik 32 gijos. Pirmiausia planuoklis įvykdo giją su didžiausia esama pirmenybe. Jei išstumiantis

režimas yra įgalintas, tai didesnio prioriteto gija, planuoklio pagalba, gali pasinaudoti sistema anksčiau už tuo metu vykdomą giją. Daugiapakopės eilės metodą sudaro eilių rinkinys, kiekvieną eilę sudaro tam tikras gijų, turinčių tokį patį prioritetą, skaičius. Pagal nutylėjimą kiekviena eilė naudoja fiksuoto laiko algoritmą, siekiant padalinti centrinio procesoriaus laiką lygiai visoms eilėje esančioms gijoms. Šioms gijoms suteikiama pirmenybė, pagrįsta kai kuriomis individualiomis gijos charakteristikomis. Visos šios eilės yra planuojamos įprasta pirmenybės eile, kai iš pradžių vykdoma didesnio prioriteto eilė.



Šaltinis: sudaryta pagal (Wu, Chen, 2011)

9 pav. eCos RTOS struktūra ir sluoksniai

Fig. 9. *eCos RTOS structure and layers*

Siekiant užtikrinti, kad sistema dirbtų mažiausiu gaišties laiku, „eCos“ naudoja padalinto pertraukimo tvarkymo mechanizmą. Viena dalis yra pertraukimo apdorojimo paprogramė (*angl. Interupt service routine (ISR)*), kita dalis - atidėto apdorojimo paprogramė (*angl. Deferred Service Routine (DSR)*). Taip užtrunkama mažiausiai laiko pertraukimo apdorojimo procedūroje, tokiu būdu sumažinant pertraukimo gaišties laiką. Jeigu pertraukimo apdorojimo procedūra yra trumpa, dažniausiai DSR nevykdoma. Be to, ši technika leidžia DSR vykdyti tuo metu, kai pertraukimo įgalinimo registras yra įjungtas, kai didesnio prioriteto pertraukimai nėra išjungti ir gali būti vykdomi. Norint pradėti DSR vykdymą, kai pertraukimai yra įgalinti, pertraukimo apdorojimo procedūra turi užtikrinti, jog tie patys pertraukimai

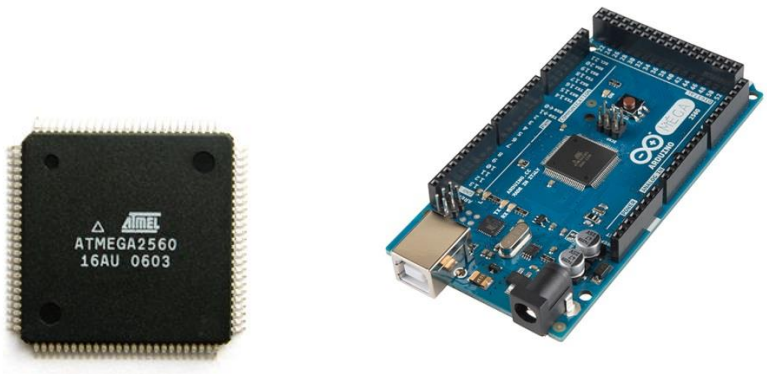
nepasikartos tol, kol atidėto apdorojimo procedūra nebus įvykdyta. Šio metodo problema tokia, kad programuotojas nežino, kiek vietos skirti kiekvienai gijai, nes „eCos“ nenurodo gijų dydžio. Taigi, vykdant atidėto apdorojimo procedūrą, įvykus kitam pertraukimui aptarnavimo metu, atidėto apdorojimo procedūra arba pertraukiamo apdorojimo procedūra gali likti be vietos.

„eCos“ branduolys muteksus ir semaforus naudoja kartu su kitu komunikavimo mechanizmu, pavyzdžiui, vėliavėlėmis ir eilėmis, norėdamas atlikti gijų sinchronizavimą.

„eCos“ naudoja pirmenybės lubas ir pirmenybės paveldėjimą, siekdama išvengti pirmenybės inversijos. „eCos“ savo pirmenybės inversijos apsaugos protokolų pakete įdiegia „paprastą“ algoritmą, kuris yra greitas ir deterministinis bei sumažina kodų dydžius, tačiau gali būti naudojamas daugiapakopės eilės planavime

3.4. Eksperimentinio tyrimo taikant realaus laiko operacines sistemas nedidelio našumo įterptinėms sistemoms rezultatai

Atsižvelgiant į kuriamai nedidelio našumo įterptinei sistemai užsibrėžtus reikalavimus (procesoriaus greitis iki 16 MIPS, programinės atminties (angl. *flash*) kiekis iki 2560 Kb, duomenų atminties kiekis iki 8KB), RTOS palyginimui buvo pasirinktas Atmel firmos gaminamas mikrovaldiklis ATmega2560 (12 pav. a).



a) Atmega2560 mikrovaldiklis b) Arduino MEGA valdiklio plokštė

Šaltiniai: (Atmel Corporation, Arduino, 2015)

10 pav. *Eksperimentui vykdyti pasirinkta nedidelio našumo įterptinė sistema*

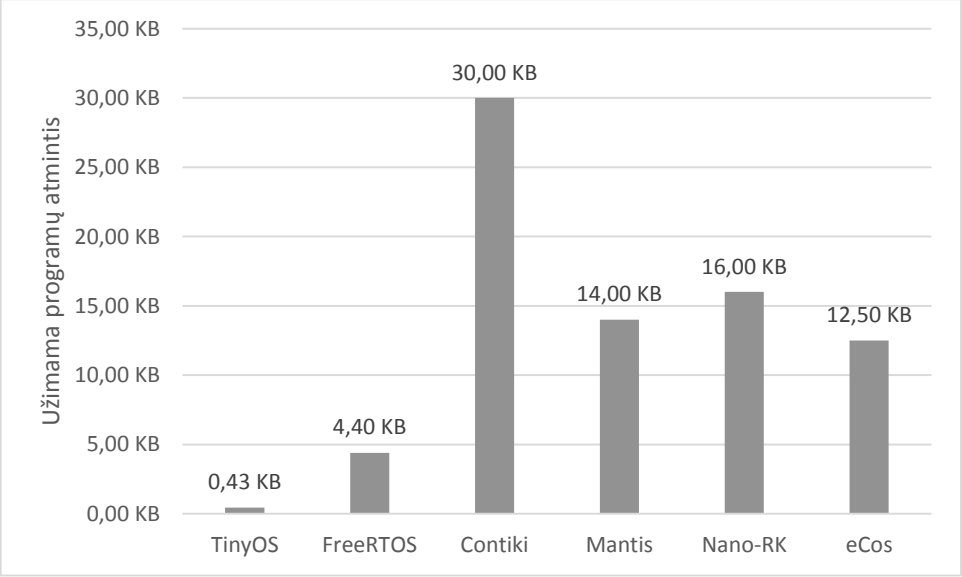
Fig. 10. *a) Atmega2560 microcontroler b) Arduino MEGA controller board*

Tyrimui naudotas valdiklis RISC architektūros, valdiklio skaičiavimo greitis 16 MIPS prie 16mhz taktinio dažnio, programų atminties tipas - flash, programų atminties kiekis - 2560

Kb, duomenų atminties tipas - EEPROM, duomenų atminties kiekis - 8 Kb, pastoviosios atminties tipas - EEPROM, atminties kiekis - 4096 b. Valdiklis turi 6 laikmačius, RTC, 32 išorinius pertraukimus, 86 įvedimo/išvedimo uostus (Atmel, 2014). Valdiklio prijungimui naudota Arduino MEGA valdiklio plokštė (8 pav. b), turinti 16 mhz kvarcinį rezonatorių, kuris užtikrina maksimalų procesoriaus darbo greitį. Valdiklio plokštė maitinama 5V įtampa iš USB sąsajos.

Tyrimo metu buvo ištrinta valdiklyje esanti Arduino programinė įranga ir ISP jungties pagalba į mikrovaldiklį įkelta kiekviena tiriamoji RTOS. Prieš įkeliant, kiekviena RTOS buvo sukompiliuota būtent šio tipo 8 skilčių mikrovaldikliui ir pridėtos papildomos f-jos, reikalingos RTOS parametrų išmatavimui.

Pirmiausiai buvo matuojama pačios operacinės sistemos užimama programų atminties vieta (angl. *flash*), kurią užpildo operacinės sistemos branduolys (angl. *kernel*), programų leistuvas (angl. *program loader*), bibliotekos ir tvarkyklės. Sukompiliavus operacines sistemas atlikti matavimai ir, iš jų paleidžiamojo failo atskyrus programų atmintį, išmatuota užimama vieta. Tyrimo rezultatai pateikti 13 paveiksle.



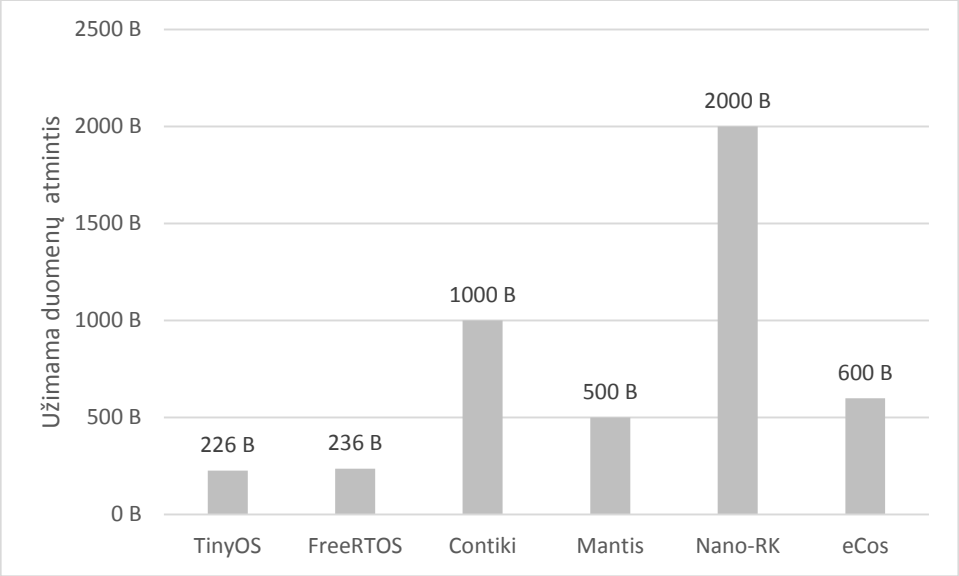
11 pav. *RTOS palyginimas pagal užimamą programų atmintį*

Fig. 11. *Occupied program memory by evaluated RTOS*

Mažiausiai programų atminties dėl savo paprastos architektūros užėmė TinyOS realaus laiko operacinė sistema – tik 432 b, šiek tiek daugiau vietos užėmė FreeRTOS operacinė sistema – 4,4 Kb. Daugiausia dėl papildomų bibliotekų užėmė Contaki operacinė

sistema – 30 Kb, tačiau nei viena tirtoji RTOS neviršijo mikrovaldiklyje esančios programų atminties vietos.

Tiriant RTOS sistemos užimamą duomenų atminties vietą (valdiklyje SRAM tipo atmintis) buvo paleistos RTOS ir išmatuotas rezervuotas atminties kiekis. Rezultatai pateikti 14 paveiksle.



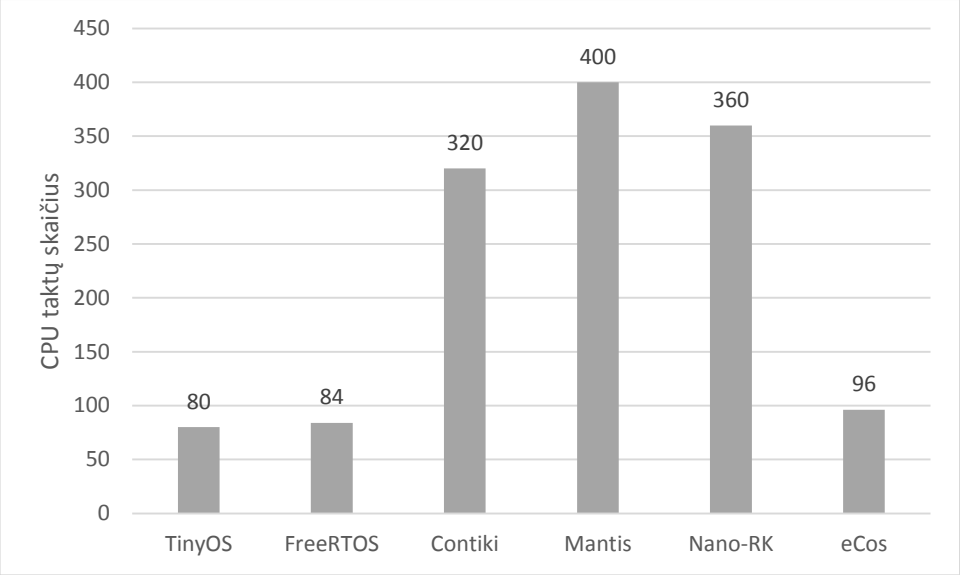
12 pav. *RTOS palyginimas pagal užimamą duomenų atminties vietą*

Fig. 12. *Occupied data memory by RTOS*

Mažiausiai duomenų atminties vėlgi užėmė TinyOS ir FreeRTOS operacinės sistemos, atitinkamai 226b ir 236b, o daugiausia vietos užėmė Nano-RK operacinė sistema – 2Kb, t. y., ketvirtį viso mikrovaldiklio duomenų atminties kiekio.

Gijos perjungimui reikalingas procesoriaus taktų skaičius buvo išmatuotas kiekvienoje operacinėje sistemoje sukūrus po du procesus ir panaudojus aparatūrinį laikmatį, esantį Amtega 2580 mikrovaldiklyje su vienetui lygiu taktinio dažnio daugikliu. Pirmasis procesas, baigdamas darbą, paleidžia laikmatį, o antrasis procesas, pradėdamas darbą, jį sustabdo. Laikmačio skaitiklio reikšmė lygi praėjusių taktų skaičiui, perjungiant vykdymo gijas. Gautieji rezultatai pateikti 11 paveiksle.

Gijų perjungimui mažiausio procesoriaus taktų skaičiaus vėlgi prireikė TinyOS ir FreeRTOS operacinėms sistemoms, atitinkamai 80 ir 84 taktų, blogesnius rezultatus parodė Contiki, Mantis ir Nano-RK operacinės sistemos: 320-400 procesoriaus taktų. Kadangi valdiklio taktinis dažnis buvo 16mhz, tai gijos perjungimui TinyOS ir FreeRTOS užtruko tik apie 5µs.



13 pav. RTOS palyginimas pagal CPU taktų skaičius reikalingų gijos perjungimui

Fig. 13. *Processor clock count required for switching between threads*

Atliekant tolimesnę RTOS sistemų lyginamąją analizę buvo nagrinėjami šie kriterijai: daugiaprograminis apdorojimas, tinklo protokolų palaikymas, programavimo kalbos palaikymas, platinimo licencija, dokumentacija, naujausia versija ir išleidimo data.

4 lentelė. RTOS sistemų palyginimas

RTOS	**Daugiaprograminis apdorojimas**	**Palaikomi tinkle protokolai**	**Programavimo kalba**	**Licencija**	**Dokumentacija**
TinyOS	Grindžiamas įvykiais	nėra	NesC	Nemokama	Funkcijų ir API aprašas
FreeRTOS	Ciklinis aptarnavimas, išstumiantis	TCP/IP, Ipv6	C	Nemokama	Išsami dokumentacija
Contiki	Grindžiamas įvykiais	TCP/IP, Ipv6	C	Nemokama	Išsami dokumentacija
Mantis	Prioritetinis	nėra	C	Nemokama	Funkcijų ir API aprašas
Nano-RK	Prioritetinis	TCP/IP, Ipv6	C	Mokama naudojant komerciniais tikslais	Funkcijų ir API aprašas
eCos	FIFO, ciklinis aptarnavimas, BitMAp	TCP/IP, IPV6, FTP, SNMP, DNS, DHCP, HTTP, SNTP	C	Nemokama	Išsami dokumentacija

Gauti analizės rezultatai, pateikti 5 lentelėje, parodė, jog visos nagrinėtos realaus laiko operacinės sistemos, išskyrus TinyOS, programavimui naudoja C programavimo kalbą. TinyOS programavimui naudoja būtent jai sukurtą C kalbos praplėtimą NesC (angl. *network embedded systems C*). Beveik visos (išskyrus TinyOS ir Mantis) nagrinėjamos RTOS turėjo TCP/IP protokolo palaikymą, įgyvendintą pačioje realaus laiko operacinėje sistemoje. Šis palaikymas kuriamajai agentinei sistemai reikalingas tam, kad ją sudarančios įterptinės sistemos galėtų komunikuoti tiek tarpusavyje, tiek su išorinėmis sistemomis. TCP/IP palaikymas gali būti realizuotas ir vartotojo srityje, tačiau tai papildomai reikalautų programų ir duomenų vietos. Išsamiausiai parengtas dokumentacijas turėjo FreeRTOS, Contiki ir eCos, likusios realaus laiko operacinės sistemos turėjo tik jų teikiamų funkcijų, struktūrų bei branduolio aprašus. Dažniausiai atnaujinama (išleidžiamos oficialios versijos) buvo FreeRTOS operacinė sistema. Nors eCos ir Nano-RK oficialios versijos išleistos pakankamai seniai, tačiau jų programinis kodas yra pateiktas internete bei gali būti koreguojamas kiekvieno registruoto vartotojo, todėl dažnai randamos ir jų neoficialios versijos.

3.5. Trečio skyriaus išvados

1. Palyginus RTOS savybes ir eksperimentiškai išmatavus jų greitaveiką nedidelio našumo įterptinėje sistemoje (RTOS buvo pritaikytos 8 skilčių, nedidelio našumo mikrovaldikliui ATmega2560) galima teigti, jog tinkamiausia DAS kūrimui nedidelio našumo įterptinėms sistemoms yra FreeRTOS. Eksperimentinio tyrimo metu nustatyta, jog FreeRTOS operacinė sistema 8 skilčių nedidelio našumo įterptinėje sistemoje užimanti 4Kb vietos (1,5%), 236 b (3%) duomenų atminties, naudoja 64 b vienai užduočiai valdyti, jai reikalingi 84 procesoriaus taktai užduotims perjungti, naudojanti prioritetinį užduočių perjungimą, todėl lieka pakankamai resursų DAS realizavimui. FreeRTOS, ji kaip ir TinyOS užima nedaug programinės ir duomenų atminties vietos, tačiau turi realizuotus TCP/IP protokolus kurių realizavimas taikomųjų programų srityje TinyOS sistemoje pareikalautų papildomų resursų, taip pat FreeRTOS turi žymiai išsamesnę dokumentaciją ir yra dažniau atnaujinama.

4. DAUGIAAGENTINĖS SISTEMOS KŪRIMO METODIKA NEDIDELIO NAŠUMO ĮTERPTINĖMS SISTEMOMS INTEGRUOTI

Skyriuje pristatoma metodika, grindžiama transformacijos metodu, kurio metu kuriama daugiaagentinė sistema ir projektavimo bei realizacijos etapuose pereinama nuo valdiklio agento sistemos kūrimo prie įterptinės sistemos techninės ir programinės įrangos realizacijos. Pristatomas DAS kūrimo principais grindžiamas įterptinių sistemų kūrimo ciklas apimantis programinės ir techninės įrangos integravimo lygmenis. Projektuojant agentų užduotis, jos priskiriamos įterptinėms sistemoms (mikrovaldikliams) techninės įrangos lygmeniu ir koordinuojamos realaus laiko operacinės sistemos (RTOS) programinės įrangos lygmeniu. DAS agentus galima suskirstyti į keletą smulkesnių sistemų pagal, valdymo įtaisus ir funkcijas. Galimi trys projektuojamos sistemos išskirstymo būdai:

- Kiekviena įterptinė sistema veikia kaip atskiras agentas ir kiekvienas agentas atsako už vienos įterptinės sistemos valdymą ir kontrolę, visi įeinantys ir išeinantys duomenys apjungti į vieną taisyklių rinkinį. Tai gali sudaryti didelę taisyklių bazę, tačiau akivaizdu, kad kai kurie kontrolės duomenys gali būti nereikšmingi. Šis metodas yra decentralizuotas kontrolės atžvilgiu.
- Visa sistema klasifikuojama keletu funkcinių modulių. Pavyzdžiui tokių kaip temperatūros reguliavimo, apšvietimo ar kt. Kiekvieną sistemos funkciją valdo atskiras vienodo lygio agentas.
- Visa sistema gali būti išskirstyta tam tikrai hierarchiniais lygiais. Kiekvieną įterptinę sistemą valdo atskiri agentai, o juos pagal funkcinius modulius valdo agentai atsakingi už tam tikrą funkciją. Visą sistemą valdo aukščiausios hierarchijos lygyje esantis valdymo agentas.

Analizuojant ir sprendžiant sudėtingas problemas specialus vaidmuo yra skiriamas agento ir daugiaagentinės sistemos DAS kūrimo etapams. Agentas reiškia abstraktų subjektą, kuris gali išspręsti tam tikrą arba dalinę problemą. Keli agentai gali būti integruoti į bendrą sistemą, kurioje veikdami kartu gali susidoroti su sudėtingesnėmis problemomis. Šie problemų skaidymo ir integravimo metodai plačiai nagrinėjami (Carrasco, 2005; Kamal 2009; Marcus 2007) darbuose. Agentų sąveikos metodais grindžiamas požiūris yra plačiai naudojamas skirtingose programinės įrangos, gamybinių sistemų ir mechatronikos srityse (Kamal, 2009; Marcus, 2007), ne tik organizuojant struktūrą aukštesniame abstrakcijos lygmenyje, bet ir kuriant platformą žemesniame (fizinių aparatūrinių dalių) lygmenyje. Tikimasi, jog išskirstytos

daugiaagentinės sistemos veikimo principais paremtas požiūris, padės efektyviai sąveikauti pažangiems agentams, fizinių prietaisų valdymui ir taip sukurti pažangias automatines sistemas išplėtotoje techninėje struktūroje (Braubach, 2005).

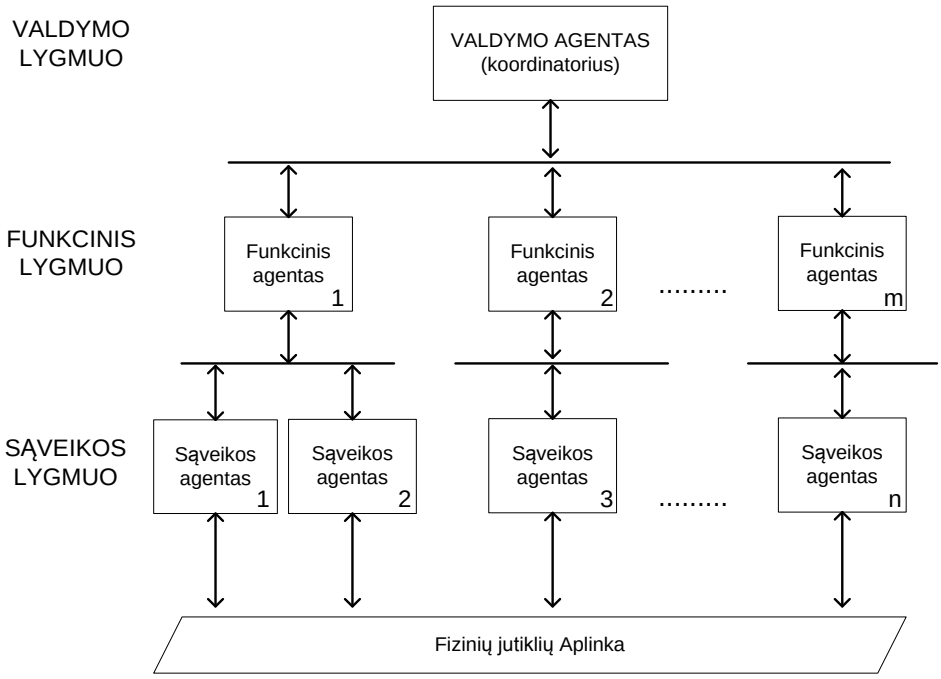
Nustatant valdiklio ir agento veikimo principus darbe savo darbe (Breemen, 2008) įveda atskiro agento tipą pavadintą valdiklio agentu. Valdiklio agentas apima visą tam tikros valdymo problemos informaciją ir valdymo metodus. Tokia įterptinių komponentų įgyvendinimo sistema yra aprašyta, siekiant sukurti ir įgyvendinti hierarchinės struktūros įterptinę sistemą, kuri sudaryta iš nevienalyčių valdymo algoritmų rinkinio. Ning remiantis šiuo požiūriu 2010 metais, pristatė įterptinę programinės įrangos sistemą – teleoperacinį mobilų robotą, kurio veikimas buvo paremtas daugiaagentinės sistemos metodais. robotą.

Efektyvios kūrimo priemonės yra labai svarbios įterptinių sistemos plėtrai bei pažangiai, daugiaagentinių sistemų metodais paremtai, kūrimo metodikai. Įprastinį sprendimą tokioms valdymo sistemoms paprastai sudaro bendrosios paskirties kompiuteris su keliomis funkcinėmis valdymo plokštėmis techninės įrangos lygmenyje ir operacine sistema („Windows“, ar k. t.) programinės įrangos lygmenyje. Tačiau šis universalus sprendimas turi keletą neišvengiamų trūkumų dėl realaus laiko operacijų atlikimo, sąnaudų ir suvartojamos elektros energijos (Jean-Paul J., 2010). Šią problemą gali išspręsti įterptinės sistemos (ARM, DSP) kurios tampa kompiuterinės pramonės svarbiausia dalimi ir pirmuoju pasirinkimu, kai valdiklis kuriamas mechatronikos reikmėms (Jean-Paul J., 2010).

Valdiklio agentas gali būti laikomas savarankišku vienetu valdymo sistemoje. Darbe bus laikomasi pagrindinės nuostato, kad kiekvienas įterptinis valdiklis atitiks atskirą agentą. Tai gali būti fizinis valdiklis arba valdymo programa. Orientuojantis į funkcijos tikslus (tikėjimus), valdiklio agentų grupė gali suprasti jų pačių veikimo algoritmus bei veikimo procesą (elgesį) ir bendradarbiauti siekdami įgyvendinti bendrą tikslą, komunikuodami ir sąveikaudami su kitais agentais (koordinavimo mechanizmais). Daugiaagentinė sistema turi palyginti laisvą ir nepriklausomą struktūrą, o agentų bendradarbiavimas gali atlikti daugybę sudėtingų užduočių, kurių nesugeba atlikti individualūs agentas.

4.1. Hierarchinis daugiaagentinės sistemos projektavimas ir konstravimas

Sistemos kūrimas prasideda nuo funkcijų skaidymo ir valdiklio agentų skaičiaus nustatymo (Hagras *et al.*, 2004). Agentų organizacija modeliuojama hierarchiniais lygmenimis pateikiama 14 paveiksle.

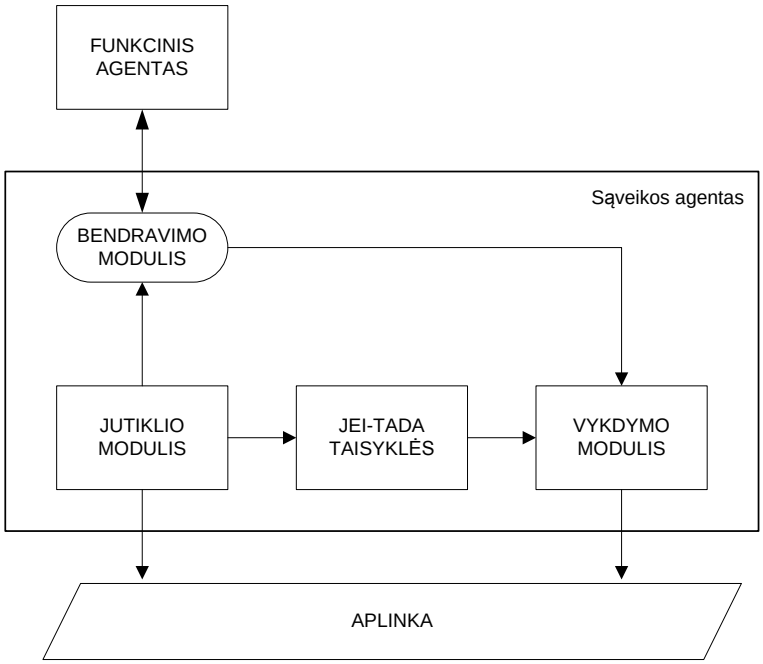


14 pav. DAS komponentų sąveikos abstrakti struktūra

Fig. 14. *DAS struktural levels*

Sąveikos lygmenyje agentai sąveikauja su fiziniais jutikliais ir valdikliais. Funkciniame lygmenyje kiekvienas agentas atlieka tam tikrą funkciją. Kiekvienas funkcinis modulis yra agentas, kuris valdo atitinkamus žemiausiojo lygio agentus. Be to, agentas gali nustatyti elgesį ir pirmumą, pagerinti sistemos intelektą ir suvokti aplinkos pokyčių ar vartotojo kontrolės idėją. Aukščiausias yra valdymo lygis. Valdymo agentas valdo visą sistemą.

Sąveikos agentai, įskaitant vidinės ir išorinės aplinkos parametrų agentus perduoda surinktą informaciją funkciniams agentams ir gauna kontrolinius signalus iš funkcinių agentų aplinkos kontrolei. Agento modelis parodytas 15 pav. Jo vidinį modelį sudaro jutiklio modulis, vykdymo modulis, bendravimo modulis ir IF-THEN (Jei-Tai) modelis. Sąlygos-veiksmo taisyklės asocijuoja jutiklį su vykdymo modeliu.



Šaltinis: sudaryta pagal (Hagras H. 2004, Ostaševičiūtė L. 2009)

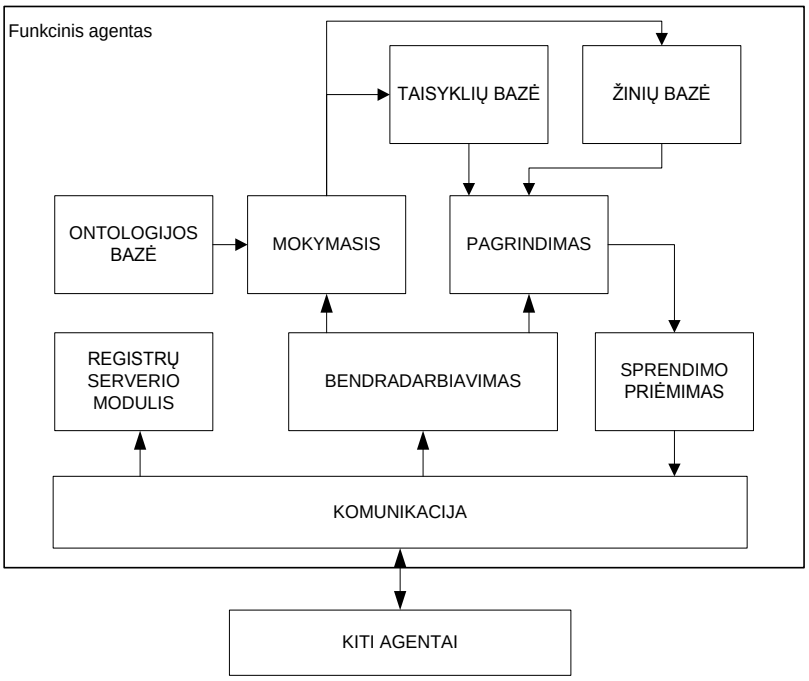
15 pav. *Struktūriniai sąveikos agento komponentai*

Fig. 15. *Structural components of interaction agent*

Ne visus sistemai funkcionuoti reikiamus sprendimus priima funkciniai agentai, kai kuriuos tiesiogiai gali priimti sąveikos agentai žemesniame lygyje tam, kad sumažinti sistemos sudėtingumą ir padidinti sistemos stabilumą. Grynai reaktyviuosius sprendimus sąveikos agentai vykdo tiesiogiai per Jei-Tada taisykles. Jei-Tada taisyklės yra tarp jungiklių ir tarp atitinkamų jungčių. Šis metodas leidžia sistemai veikti tik pagrindiniame lygyje, net jei funkcinis agentas neveikia. Iš diagramos matyti, kad valdymo modulio įėjimą sudaro sąveikos agento sprendimas ir kontrolės informacija iš funkcinio agento. Sąveikos agento sprendimas yra viršesnis už funkcinio agento, todėl vykdymo agentas pasirenka sąveikos agento sprendimą, jeigu jie konfliktuoja. Sąveikos agentas vykdo atitinkamus veiksmus, jei taisyklė pritaikoma sėkmingai. Tai reiškia, kad sąveikos agentas turi veiksmų užduotis. Priešingu atveju, jei yra komunikacijos įvykis iš funkcinio agento, tuomet jis yra vykdomas.

Funkciniai agentai atsako už tam tikrą sistemos vykdomąsias funkcijas. Dažniausiai vidinė funkcinių agentų struktūra (16 pav.) yra tokia pati, o jų funkcijos skiriasi. Registrų serverio modulis atsako už tinklo komunikaciją ir registruoja tinklo ryšio registre. Jis nustato tinklo maršrutų sąrašus belaidžiame tinkle, kai jis yra kuriamas. Agentai, susiję su funkcinio agento valdoma funkcija, tokie kaip sąsajos ar valdymo agentas, privalo sudaryti maršrutus su juo. Tinklo kūrimo procese kai kurie agentai gali būti pridėti, o kiti pašalinti, siekiant padidinti

sistemos lankstumą. Informacija, gaunama iš sąveikos agentų, dažniausiai susideda iš aplinkos parametrų, išorinių aplinkos parametrų, prietaisų būsenos ir pan.



Šaltinis: sudaryta pagal (Hagras et al., 2004)

16 pav. *Struktūriniai funkcinio agento komponentai*

Fig. 16. *Structural components of functional agent*

Dėl nuolatinių pokyčių aplinkoje ir naudotojų prioritetų, funkcinis agentas yra evoliucinė funkcija ir neretai naudojanti tam tikrus dirbtinio intelekto metodus (tokius kaip neraiškiąją logiką ar kt.). Kai atsiranda pokyčiai aplinkoje pakeičiama tiek taisyklių bazė, tiek žinių bazė, siekiant pagerinti sprendimų priėmimo funkciją.

Valdymo agentas. Pritaikydamas žinių ir gebėjimų bazę valdymo agentas aukščiausiu lygmeniu valdo visą sistemą. Agentas patvirtina bendras užduotis ir platina užduotis skirtingiems funkciniams agentams. Jis yra atsakingas už užduočių paskirstymą bendradarbiavimą ir konfliktinių situacijų pašalinimą (kai viena užduotis prieštarauja kitai). Agentas turi automatinę valdymo funkciją, kuri palaiko sistemos veikimą ir nagrinėja sistemos įvykius.

Be to, ji turi nepriklausomo planavimo galimybę. Vidinė struktūra gali būti išskirstyta į tris sluoksnius. Aukščiausias sluoksnis, skirtas laikyti visos sistemos informaciją ir, remiantis ontologijų baze, žinių baze ir duomenų baze, perduoti atitinkamas žinias kitiems agentams. Duomenų bazė apima aplinkos informacinius duomenis, prietaisų duomenis ir vartotojo

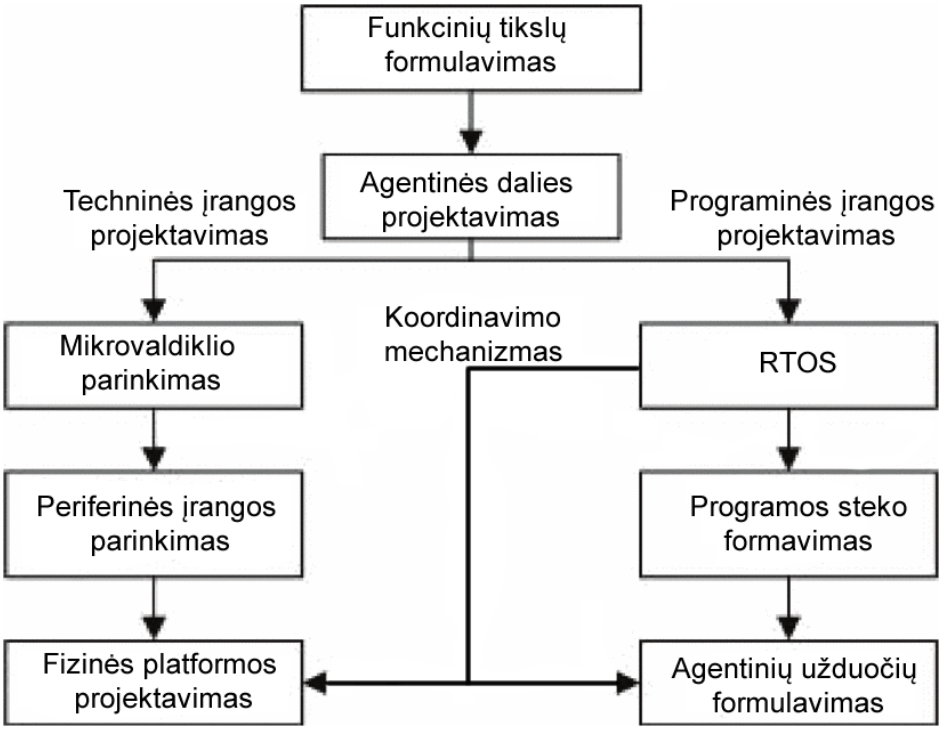
informacijos duomenis. Žinių bazę sudaro globalios žinios, užduočių žinios, bendradarbiavimo žinios, konfliktų pašalinimo žinios ir t.t.

Vidurinį sluoksnį sudaro bendradarbiavimo modulis, konfliktų pašalinimo modulis ir užduočių plano modulis. Bendradarbiavimo modulis paskirsto bendradarbiavimo politiką tarp agentų, atliekančių vieną užduotį kartu. Užduočių plano modulis patvirtina funkcinių agentų įvykdytas užduotis ir paskirsto naujas užduotis atitinkamiems agentams. Konfliktų sprendimo modulis pašalina konfliktus, kai konfliktai įvyko, ar gali įvykti. Žemiausias sluoksnis susijęs su komunikacijos moduliu, jį sudaro registrų serverio modulis, automatikos valdymo modulis, maketo modulis, užduočių veiksmų modulis ir būsenų serverio modulis.

Registrų serverio modulis skirtas informacijos valdymui. Kiekvienas agentas bevieliame tinkle turi unikalų identifikatorių. Jis priskiria kiekvienam agentui identifikatorių ir valdo visus maršrutus intelektinėje aplinkoje. Automatikos valdymo modulis palaiko normalų sistemos veikimą ir reaguoja į netikėtus įvykius. Užduočių veiksmų modulis vykdo užduotis iš užduočių plano modulio ir paskirsto užduotis atitinkamiems funkciniams agentams. Sąveikos agentai perduoda perprantamą informaciją valdymo agentui, tarpininkaujant funkciniam agentui. Taigi, valdymo agento būsenų serverio modulis turi informaciją apie aplinką ir prietaisus.

4.2. Daugiaagentinių sistemų kūrimo metodų taikymo žingsniai įterptinių sistemų integracijai

Įterptinėse sistemose, pritaikius valdiklio agentų mąstymą ir koncepciją, galima sukurti gan efektyvią ir aiškios sąveikos struktūros sistemą (vienas agentas - vienas mikrovaldiklis), grindžiamą daugiaagentinių sistemų kūrimo metodikos žingsniais. Siekiant sukurti pilnavertę fizinę platformą mikrovaldikliai gali naudoja ribotą aibę luste esančių įrenginių ir prie jų prijungtų periferinių įrenginių. Agentų užduotys yra vykdomos kaip programos su stekais realaus laiko operacinėje sistemoje, o tarp jų esantis koordinavimo mechanizmas, gali būti programuojami panaudojant taikomųjų programų sąsają (API), kurias teikia RTOS. Visi įterptinių sistemų integruotos sistemos kūrimo etapai pavaizduoti 17 paveiksle.



17 pav. ISIS sistemos komponavimas

Fig. 17. *Development phases of ISIS system*

4.3. Techninės įrangos pasirinkimas valdiklio agentams

Valdiklio agentams turintiems skirtingus tikslus gali reikėti įvairios techninės įrangos, kad jie galėtų atlikti atitinkamus veiksmus, pavyzdžiui, mikrovaldikliuose esančių įrenginių, kad jie vykdytų tam tikras komandas ir specifinių periferinių įrenginių, kad būtų išplėstos sistemos funkcijos. Fiziniai prievadai šiems, skirtinguose mikrovaldikliuose esantiems valdiklio agentams, suteikia tarpusavio sąveikos priemones. Išorinė atmintis gali būti lentos bendravimo sistemų realizavimo forma, skirtinguose mikrovaldikliuose esančių valdiklio agentų komunikacijai. Šių agentų įgyvendinimas techninės įrangos lygiu turi ne tik panašumą bet taip pat ir funkcijų bei struktūros įvairovę, todėl agentai gali būti klasifikuojami į skirtingas agentų klases, turinčias įvestį, išvestį, komunikavimo įrangą ir atmintį. Taigi, kuriant valdymo sistemų techninės įrangos platformą ir organizuojant techninės įrangos išteklius, gali būti laikomasi modulinio projektavimo principų.

Kai kuriems valdiklio agentams gali reikėti tik mikrovaldiklio procesoriaus, tam kad jie galėtų vykdyti algoritmus ir atlikti skaičiavimus. Tokio tipo agentai gauna informaciją (arba negauna) iš kitų agentų ir atlikę skaičiavimus tiesiog perduoda nurodymus kitiems agentams (patys neatlieka jokių matavimų ir valdymo funkcijų). Kito tipo valdiklio agentai gali gauti

informaciją iš skirtingų jutiklių prijungtų prie mikrovaldiklio (tiek analoginių tiek ir skaitmenini), gali naudoti analoginės ir skaitmeninės išvesties jungtis ir periferinius įrenginius tam, kad galėtų valdyti skirtingus aktyvatorius. Kai kurie valdiklio agentai naudoja įvairius komunikavimo prievadus ir magistrales, pavyzdžiui nuoseklųjį (UART) prievadą ir nuosekliąją duomenų perdavimo sąsają (SPI), integrinių grandynų sąsają (I2C), transporto priemonių duomenų perdavimo tinklo magistralę (CAN) ir k. t. siekiant įvykdyti duomenų perdavimą tarp skirtingų mikroprocesorių. Taigi, šie agentai yra išdėstyti skirtingų struktūrų lygiuose ir gali naudoti įvairius techninės įrangos išteklius, atsižvelgiant į jų funkcinius tikslus.

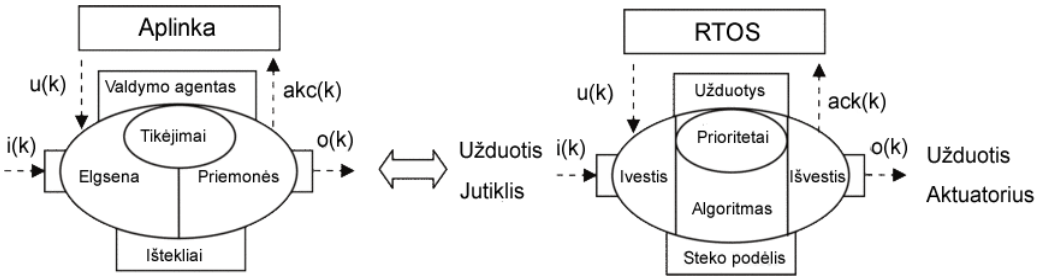
4.4. Programinės įrangos kūrimas valdiklio agentams

Visa valdymo sistema suskirstoma į keletą valdiklio agentų, atsižvelgiant į funkcinius šios sistemos tikslus. Valdiklio agentai suskirstomi į skirtingus sistemos sluoksnius: vykdymo, bendradarbiavimo ir organizavimo. Kai kurie stipriai tarpusavyje susiję agentai gali būti sujungti į vieną valdymo agentą turintį bendrus valdymo tikslus ir vietinį koordinavimo mechanizmu. Pagrindinis agentas sukuria lentos bendravimo sistemą (angl. *blackboard*), kurios pagalba dalijamasi informacija ir pasiekiama sąveika tarp visų daugiaagentinės sistemos dalyvių. Daugelio agentų struktūra yra ypač tinkama lygiagrečiam užduočių apdorojimui ir gali sumažinti resursus naudojamus dideliam duomenų kiekio perdavimui. Tokioje struktūroje visi agentai gali dirbti nepriklausomai skirtinguose informacijos lygiuose. Kiti komunikacijos būdai, tokie kaip, taškas į tašką (angl. *Point-to-point*) ar transliacija (angl. *broadcasting*) taip pat gali būti naudojami, siekiant pagerinti realaus laiko sistemos darbą tiems agentams, kurie turi specifinius komunikavimo partnerius.

Valdiklio agentus galime panaudoti įterptinėse sistemose, naudojant daugiafunkcinį RTOS branduolį programinės įrangos lygiu. RTOS užduotis yra paprasta programa, kuri sukuriamas įvaizdis, kad ji pati savyje turi centrinį procesorių (CPU). Kiekvienai užduočiai yra priskiriamas prioritetas, centrinio procesoriaus (CPU) registrai ir atminties vieta (dažniausiai steko pagrindu). Pagrindinė RTOS branduolio teikiama paslauga šiuo atveju yra turinio perjungimas (planuoklio naudojimas) ir taikomųjų programų sąsajos (API) suteikimas. Naudodama šį metodą įterptinė sistema turi realaus laiko daugiafunkcinio apdorojimo galimybę, o kiekviena RTOS užduotis sąlyginai gali būti vykdoma vienu metu (naudojant daugiaužduotiškumą). Kiekvieną užduotį galime laikyti kaip nepriklausomą pseudoprocesorių, kurį gali planuoti daugiafunkcinis RTOS branduolys. Jie taip pat gali komunikuoti tarpusavyje,

priklausomai nuo RTOS sistemos funkcijų. Šių funkcijų negalima pasiekti vien tik padidinant procesoriaus pajėgumą.

DAS atžvilgiu galime sukurti kiekvieną užduotį, paremtą valdiklio agentų modeliu programinės įrangos lygiu ir sukonstruoti jų koordinavimo ir komunikavimo mechanizmą, paremtą RTOS sistemos funkcijomis. Transformacijos modelis iš valdiklio agento į RTOS užduotis yra pavaizduotas 18 pav. Visos valdiklio agento užduotys ir kiti ištekliai yra paverčiami užduočių stekais ir buferiais, agento tikėjimas ir jo tikslų modelis – užduoties branduoliu ir jo pirmenybe, agento elgesys – užduoties algoritmais, agento priemonės – užduoties įvesties ir išvesties sąsajomis. Proceso, algoritmų ir sąsajų operacijos galiausiai yra aprašomos užduoties programiniame kode. Kodas, stekas ir pirmenybė yra trys pagrindiniai elementai, reikalingi užduočiai sukurti.



18 pav. Transformacijos schema pertvarkant valdiklio agento užduotis į RTOS užduotį

Fig. 18. *Transformation schema from controller agent to RTOS task*

Keturi interaktyvūs režimai tarp valdiklio agento ir aplinkos, įskaitant įvestį i(k), išvestį o(k), prašymą aktyvavimui u(k) ir patvirtinimą veikimui ack(i), gali būti atitinkamai realizuojami įvesties / išvesties operacijomis ir RTOS sistemos funkcijomis, norint koordinuoti skirtingas užduotis ir perduoti bendrus duomenis. Agento užduoties įvesties duomenys gali būti gaunami tiesiogiai iš jutiklių, naudojant įvesties sąsajas, arba netiesiogiai iš kitų užduočių, naudojant pranešimus arba eilės eiles. Agento užduoties išvesties duomenys gali tiesiogiai valdyti vykdomuosius įrenginius arba nusiųsti kitoms užduotims, naudojantis pranešimu. Agento užduotis gali laukti savo aktyvavimo sąlygų, pavyzdžiui kol atsilaisvins tam tikras įrenginys. Įrenginius paskirstymą vykdo semaforai.

Valdiklio agento koordinavimo mechanizmas gali būti realizuojamas bendroje atmintyje (SMP) arba RTOS sistemos funkcijomis. SMP yra bendros atminties vieta adresuojamoje atmintyje (RAM) globaliam duomenų saugojimui, tinkama lentos

komunikavimo sistemai realizuoti. Kiekviena agento užduotis turi skirtingą prieigos teisę prie bendrų duomenų bendroje atmintyje (SMP). Semaforas yra priskiriamas kiekvienai bendrų duomenų grupei, siekiant užtikrinti duomenų patikimumą ir vientisumą. Užduotis gali turėti prieigos galimybę tik tada, kai semaforas jai tai suteikia.

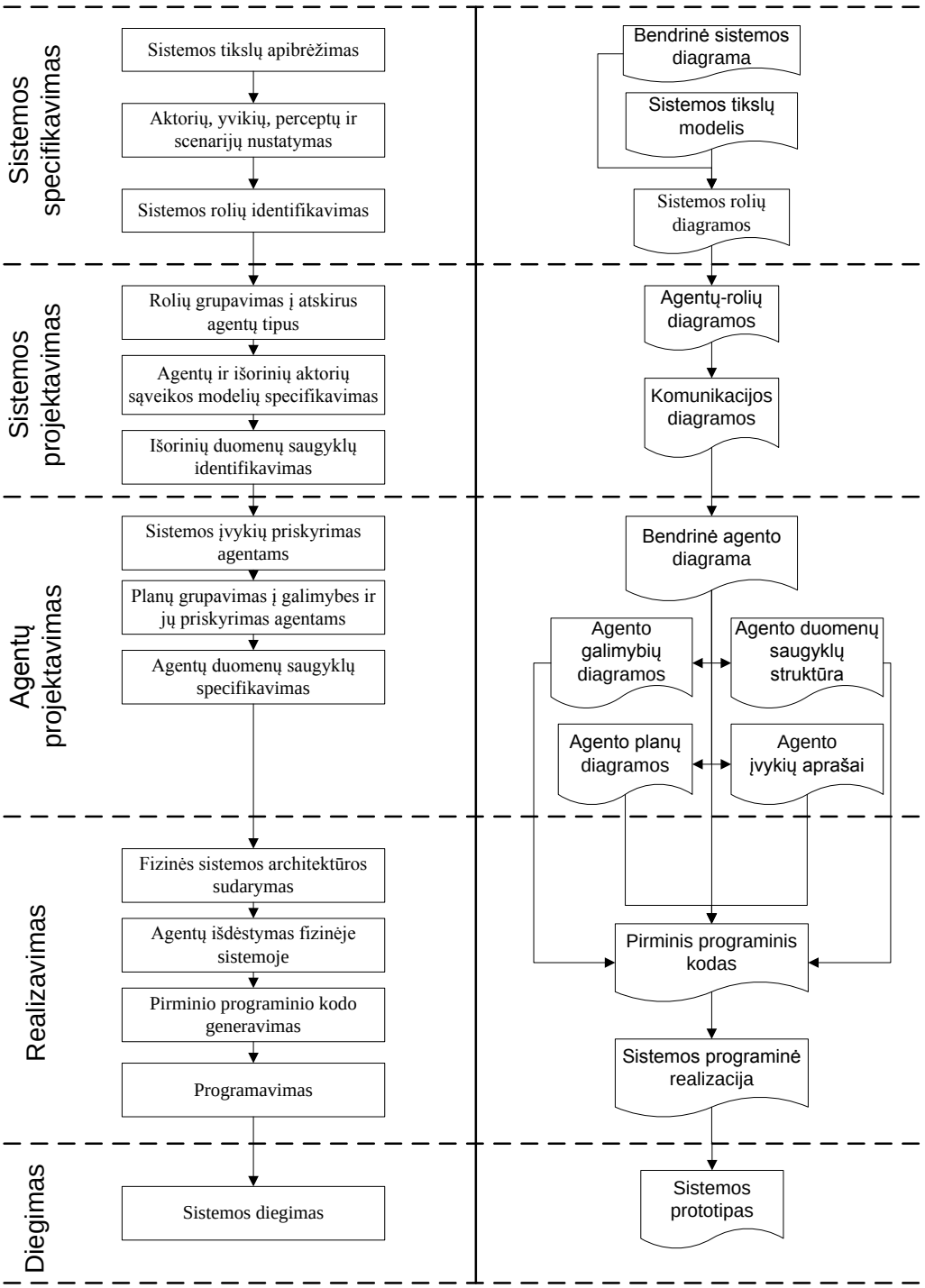
4.5. DAS skirtų nedidelio našumo įterptinėms sistemoms kūrimo metodika

Pasiūlytoji DAS, skirtų nedidelio našumo įterptinėms sistemoms, kūrimo metodika apima penkis pagrindinius etapus: sistemos specifikavimas, sistemos projektavimas, agentų projektavimas, realizavimas, diegimas. Ši metodika grindžiama prototipų kūrimu (iteracine sistemų kūrimo paradigma). Prototipas – pradinė sistemos versija, demonstruojanti sukurtosios sistemos ar projektinius sprendimus.

Sistemos specifikavimo etapas pasiūlytoje metodikoje susideda iš trijų žingsnių: sistemos tikslų apibrėžimas, aktorių, įvykių, perceptų ir scenarijų nustatymas, sistemos rolių indentifikavimas.

Sistemos kūrimas prasideda apibrėžiant jos tikslus. Pirmiausia identifikuojami pagrindiniai sistemos tikslai, t.y. aprašomos sistemos vykdomos funkcijos. Tuomet, kiekvienam iš šių tikslų yra keliamas klausimas: „Kaip šis tikslas gali būti pasiekiamas?“ Iš atsakymų aibės sudaromi potiksliai, kuriuos įgyvendinus bus pasiektas pagrindinis sistemos tikslas. Potiksliai gali būti privalomi (norint pasiekti tikslą, privaloma visus juos įgyvendinti) arba alternatyvūs (norint pasiekti tikslą, privaloma įgyvendinti bent vieną potikslį). Taip sudaromas hierarchinis sistemos tikslų modelis, kuris pavaizduojamas tikslų diagrama (angl. *goal overview diagram*), joje privalomi potiksliai sujungiami „AND“, o alternatyvūs – „OR“ valdymo elementais.

Aktorių, įvykių, perceptų ir scenarijų nustatymo žingsnyje identifikuojami aktyvūs ir pasyvūs sistemos elementai, bei jų sąveika su aplinka. Tai vykdoma dviem etapais: pirmiausia, nustatomi sistemos aktoriai ir scenarijai, kurių metu jie įtraukiami į sistemos veiklą, vėliau nustatomi veiksmai, kuriuos vykdo aktoriai bei perceptai, atsirandantys tarp atskirų aktorių, ir sistemos. Aktoriais vadinami kuriamosios sistemos vartotojai, išoriniai įrenginiai, perduodantys informaciją arba yra valdomi, taip pat ir kitos, išorinės sistemos. Perceptai – pirminė informacija, gaunama iš aplinkos. Šiame žingsnyje sudaroma bendrinė sistemos diagrama (angl. *analysis overview diagram*), kuri yra skirta parodyti sąveiką tarp kuriamos sistemos ir aplinkos.



19 pav. DAS, skirtų nedidelio našumo įterptinėms sistemoms kurti, metodika

Fig. 19. *Methodology of MAS system development for small-scale embedded systems*

Paskutiniame sistemos specifikavimo žingsnyje identifikuojamos sistemos rolės. Rolės – tai tam tikri kuriamos sistemos funkciniai gebėjimai, apibrėžiantys tai, ką kuriamos sistemos esybės gali atlikti. Šio žingsnio metu, susiję tikslai sugrupuojami į atskiras roles. Rolėms priskiriami pirmame žingsnyje identifikuoti scenarijai bei perceptai, kurie reikalingi rolės

tikslams pasiekti. Visi antrame žingsnyje nustatyti tikslai privalo būti priskirti tam tikrai rolei. Jei priskiriamą tikslą sudaro keletas potikslių, pakanka įtraukti pagrindinį tikslą taip nurodant, jog potiksliai taip pat priklausys atitinkamai rolei. Visos rolės pavaizduotos rolių diagramoje (angl. *system roles diagram*).

Sistemos projektavimo etapą sudaro trys žingsniai: rolių grupavimas į atskirus agentų tipus, agentų ir išorinių aktorių sąveikos modelių bei išorinių duomenų saugyklų specifikavimas.

Pirmajame žingsnyje identifikuojami sistemos agentų tipai. Atpažįstami skirtingų tipų agentai, kurių gali būti vienas arba keli realizacijų kuriamojoje sistemoje. Šiame etape aprašomas ne pats agentas, o kiekvieno agento tipas. Agentų identifikavimas atliekamas grupuojant ankščiau specifikuotas roles. Atliekant grupavimą, reikia vadovautis šiais principais: rolės, naudojančios tuos pačius duomenis, turėtų būti priskirtos tai pačiai grupei, geriau kurti vieno agento tipą apjungiant kelias roles, nei kiekvienai rolei priskiriant skirtingą agento tipą. Sudaroma agentų rolių diagrama (angl. *Agent-role grouping diagram*).

Kitame žingsnyje nurodoma agentų tarpusavio bei agentų ir sistemos aktorių sąveikos. Galima išskirti tris skirtingas agentų komunikacijos formas: agentai gali tiesiogiai sąveikauti su kitais agentais, siųsdami vieni kitiems žinutes; gali komunikuoti su aplinka, naudodamiesi jutikliais ir vykdikliais; sąveikauti su kitais agentais gali ir aplinkoje, paveikdami ją savo vykdikliais. Sąveika vyksta perduodant pranešimus arba vykdant tam tikrą specifikuotą protokolą. Šiame žingsnyje yra formalizuojami tokių mainų režimai. Protokolų aprašymai agentams yra išoriniai, tačiau jie dalijasi protokolų egzemplioriais. Sąveikos protokolai aprašomi AUML notacija.

Paskutiniajame šio etapo žingsnyje identifikuojamos išorinės duomenų saugyklos. Svarbu kuo detaliau aprašyti skirtingų agentų bendrai naudojamus duomenis. Pačių agentų vidiniai duomenys specifikuojami kitame etape.

Agentų projektavimo etapą sudaro trys žingsniai: sistemos įvykių priskyrimas agentams, planų grupavimas ir jų priskyrimas agentams, agentų vidinių duomenų saugyklų specifikavimas.

Pirmajame žingsnyje išskiriami kuriamosios sistemos išoriniai bei vidiniai įvykiai. Įvykiai gali atsirasti dėl išorinių aplinkos veiksnių, juos gali sugeneruoti atskiros kuriamosios sistemos arba agentai. Kiekvienas įvykis priskiriamas antrajame etape aprašytiems agentų tipams. Skirtingiems agentų tipams gali būti paskirtas tas pats įvykis, t. y. skirtingi agentai gali reaguoti į tą patį įvykį.

Antrajame žingsnyje pagal agentui paskirtas roles ir tikslus sukuriami planai. Planas – veiksmų, kuriuos turi atlikti agentas, norėdamas įgyvendinti siekiamą tikslą arba prie jo priartėti, seka. Norint pasiekti vieną tikslą, agentui gali prireikti įgyvendinti daugiau nei vieną planą. Taip pat, agentui įgyvendinus vieną planą, gali būti pasiektas daugiau nei vienas tikslas. Susiję planai grupuojami į agento galimybes (angl. *capability*). Agento galimybės – planų ir įvykių rinkinys, reikalingas tam tikro tikslo pasiekimui. Į agento galimybes grupuojami ir alternatyvūs planai, skirti to paties tikslo pasiekimui. Agento planai ir iš jų sudarytos galimybės pavaizduojamos galimybių diagramomis (angl. *capabilities diagram*).

Paskutiniajame trečio etapo žingsnyje detaliai aprašomos vidinės agento duomenų saugyklos. Jos priskiriamos agento planams, kurių įvykdymui reikalingi šioje saugykloje esantys duomenys. Atlikus trečiajame etape aprašytus žingsnius, kiekvieno agento tipui sudaroma detali agento diagrama (angl. *agent's detailed design*), joje pavaizduojami visi agento planai, galimybės, įvykiai, duomenų saugyklos, perceptai ir pranešimai, kuriuos agentas siunčia bei gauna.

Realizavimo etapą sudaro trys žingsniai: fizinės sistemos architektūros sudarymas, agentų išdėstymas fizinėje sistemoje, pirminio programinio kodo generavimas ir programavimas.

Pirmajame žingsnyje atliekamas techninės/programinės įrangos atskyrimas bei specifikavimas. Aprašomi duomenų perdavimo protokolai (USART, I2C, SPI, CAN ir k.t.), kurie bus naudojami duomenų surinkimui iš techninės įrangos ir atskirų valdiklių komunikacijai. Taip pat aprašomi techninei įrangai keliami reikalavimai: programų ir duomenų atminčių dydžiai, skaičiavimo greitaveika, įvedimo/išvedimo uostų skaičius, analoginių/skaitmeninių keitiklių kiekis, jų raiška ir kt.). Aprašoma bendra techninės įrangos architektūra, kurioje numatomi visi jutikliai ir vykdikliai, mikrovaldikliai bei kita techninė įranga.

Kitame žingsnyje atliekamas agentų išdėstymas fizinėje sistemoje. Pagal techninės įrangos aprašus, sudarytus pirmajame žingsnyje, kiekvienai įterptinei sistemai priskiriamas tam tikras agentas. Peržiūrima, ar kiekviena įterptinė sistema, kuriai priskirtas tam tikro tipo agentas, turi techninį komunikacijos įgyvendinimą su agentais, esančiais kitose įterptinėse sistemose ir turinčiais gauti/siųsti duomenis.

Trečiajame žingsnyje atliekamas pirminio programinio kodo generavimas, kuris atliekamas iš trečiajame etape sudarytų agentų aprašų (įvykių, planų, galimybių, duomenų saugyklų, bendravimo protokolų), atsižvelgiant į pirmajame žingsnyje specifikuotą techninę įrangą. Agentams, esantiems nedidelio našumo įterptinėse sistemose, generuojama C++

programavimo kalba, o agentams, esantiems galingesnėse kompiuterinėse sistemose, gali būti generuojama C++/JAVA ar kt. programavimo kalba. Sugeneruojamos agentų klasės, jų metodai, duomenų struktūros, įvykių apdorojimui skirtos funkcijos ir kt.

Ketvirtajame šio etapo žingsnyje atliekami visi programavimo darbai: suprogramuojami agentų planai, jų parinkimo metodai, reagavimo į įvykius funkcijos ir kt.

Paskutiniajame etape atliekamas sistemos diegimas ir testavimas. Pagal ketvirtajame etape sudarytą techninės įrangos aprašą konstruojamas sistemos prototipas. Į įterptines sistemas įrašomas suprogramuotas ir sukompiliuotas programinių agentų kodas.

4.6. Ketvirto skyriaus išvados

1. Išanalizavus agentinių technologijų taikymo sritis ir jų charakteristikas, buvo nustatyta, kad agentinė paradigma tinkama projektuojamai sistemai kurti, nes užtikrina aukštą abstrakcijos laipsnį, nesudėtingą sistemos plečiamumą, integralumą, efektyvų komunikacijos mechanizmą ir kitus reikalavimus, keliamus šios klasės sistemoms ir jų kūrimo procesui.
2. Apžvelgtas bei pateiktas modelis, kaip galima transformuoti suprojektuotos DAS agentus į RTOS valdomą įterptinę sistemą. Taip pat pateikiamas skirtingų tipų agentų realizavimo metodas nedidelio našumo įterptinėse sistemose.

5. EKSPERIMENTINIO TYRIMO REZULTATAI ATLIEKANT JŪROS METEOROLOGINIŲ DUOMENŲ STEBĖSENA PASIŪLYTOS DAUGIAGENTINĖS SISTEMOS PAGRINDU

Penktame skyriuje pateikiami eksperimentinių tyrimų rezultatai, kuriuose aprašyti atliktų hidrometeorologinių stebėjimų valdymo sistemos kūrimo eksperimentiniai tyrimai ir daugiagentinės sistemos veikimo analizė. Tai nedidelio našumo įterptinių sistemų integravimo pavyzdys, kurio išplėtojimui buvo pasitelkta ankstesniuose skyriuose aprašyta DAS kūrimo platforma. Skyriuje pateikiami sukurtosios hidrometeorologinių sistemų techninės ir programinės įrangos aprašai bei rezultatai, kurie parodo, jog pasiūlytos daugiaagentinių sistemų platformos taikymas jose palengvina sistemos funkcinį plečiamumą, daro sistemas adaptyvesnes, efektyvesnes ir galinčias dirbti prie ribotų darbo našumo apribojimų.

5.1. DAS metodikos taikymas jūros meteorologinių duomenų stebėsenai

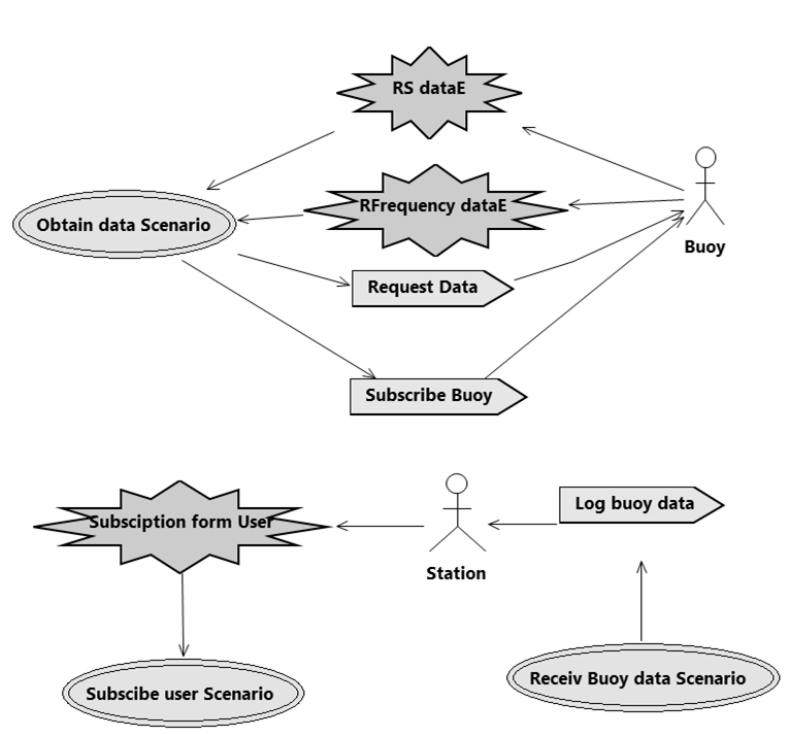
Šiuo metu yra daug įvairių tipų autonominių sistemų skirtų hidrometeorologinių ir okeanografinių stebėjimų vykdymui. Viena iš tokių sistemų naudojama Kanarų salų priekrantės stebėjimui (Bender, 2010). Šią sistemą sudaro valdymo centras, kuris kontroliuoja duomenų perdavimą ir pateikia informaciją apie svarbius socialinius ir ekonominius priekrantės sektorius, šie duomenys, gauti iš autonominių plūdurų, naudojami kontroliuoti priekrančių aplinką. Plūdurai stebi vandens temperatūrą, druskingumą, deguonies, angliavandenilių kiekį ir kitas savybes prijungus papildomus jutiklius, pavyzdžiui fluorometrą, taip pat kiekvienas plūduras turi GSM modemą, kurio pagalba persiunčiami duomenys. Plūdurai duomenis į centrinę stotį siunčia SMS žinutėmis, kurios turinį sudaro: jutiklių duomenų rinkinys, plūduro GPS padėtis ir baterijų lygis. Duomenis plūdurai siunčia kas valandą. Tačiau atlikus detalesnę duomenų analizę, nustatyta, jog toks duomenų perdavimo greitis yra nepakankamas, todėl duomenų perdavimo protokolas šio tipo sistemose turėtų būti peržiūrėtas ir pakeistas į spartesnį.

Siekiant užtikrinti didesnį hidrometeorologinių stebėjimų duomenų perdavimo patikimumą ir greitesnį jų perdavimą yra siūlomi įvairūs jutiklių tinklai (Laarhuis, 2010; Li, 2008; Collins, 2012). Taip pat siūlomos kitokios duomenų perdavimo sistemos, kurios duomenis iš jutiklių mazgų (Collins, 2012) perduoda per tarpines stotis esančias jūrinėse platformose, komerciniuose laivuose ir kituose jūriniuose objektuose. Šios sistemos neretai naudoja akustinius mesh tinklus (Laarhuis, 2010). Toks siūlomas sistemos modelis išspręstų

daug duomenų rinkimo problemų, tačiau tam reikalingos lanksčios sistemos, kurias būtų galima nesunkiai integruoti į įvairius jūrinius objektus. Ši problema gali būti išspręsta sukuriant autonominę daugiaagentinę sistemą, kuri galėtų vykdyti gautų duomenų apdorojimą ir pirminę analizę priklausomai nuo esamos situacijos.

5.2. Hidrometeorologinių duomenų rinkimo sistemos specifikavimas

Bendrinė sistemos diagrama (angl. *Analysis Overview Diagram)* skirta parodyti sąveiką tarp kuriamos sistemos ir aplinkos. Šiame abstrakčiame lygyje svarbu nusistatyti sistemos aktorius, perceptus ir įvykius įtakojančius daugiaagentinės sistemos veikimą. Diagrama sudaroma dviem etapais: pirma nustatomi aktoriai ir scenarijai kurių metu jie įtraukiami į sistemos veiklą, antra nustatomi ir aprašomi veiksmai ir perceptus tarp atskirų aktorių ir sistemos. Bendrinė HDRS diagrama pateikiama 20 pav.



20 pav. *Bendrinė hidrometeorologinių duomenų rinkimo sistemos diagrama*

Fig. 20. *Analysis overview diagram of hydrometeorological data collection system*

Sistemos aktoriai yra vartotojai arba išoriniai įrenginiai (arba kitos sistemos), kurie yra susiję su kuriama sistema. Kuriamąją HDRS daugiaagentinę sistemą sudaro du aktoriai:

„Buoy“ – apibrėžia plūduro techninę įrangą (jutiklius), kurie nuskaitytus duomenis perduoda HDRS daugiaagentinei sistemai;

„Station“ – stebėjimų stotis, kuriai perduodami duomenys ir reikalui esant nuotoliniu būdu valdomi plūdurai.

Toliau aprašomi scenarijai (procesai, kuriuos sistema naudoja) apdoroti gautus perceptus ir atlikti tam tikrus veiksmus. HDRS identifikuojami trys aukščiausio lygio scenarijai:

„Obtain data scenario“ – scenarijus, kurio metu daugiaagentinė sistema iš plūduro techninės įrangos gauna išmatuotus parametrus;

„Subscribe user scenario“ – scenarijus, kurio metu sukuriamas prisijungimas hidrometeorologinių stebėjimų stotyje skirtas vartotojui (arba kitai sistemai) gauti stebėjimo duomenis;

„Receiv buoy data scenario“ – scenarijus, kurio metu iš hidrometeorologinių stebėjimo plūdurų išmatuoti ir apdoroti duomenys perduodami stebėjimų stočiai.

Perceptai aprašo įvairią informaciją, kuri patenka į sistemą iš aplinkos. HDRS sistemoje iš viso yra trys tokie perceptai:

„RS dataE“ – duomenys gauti iš aparatūrinių sensorių;

„RFrequency dataE“ – nuskaitymo dažnumo pasikeitimo informacija;

„Subscription from user“ – vartotojo (arba kitos sistemos) informacija.

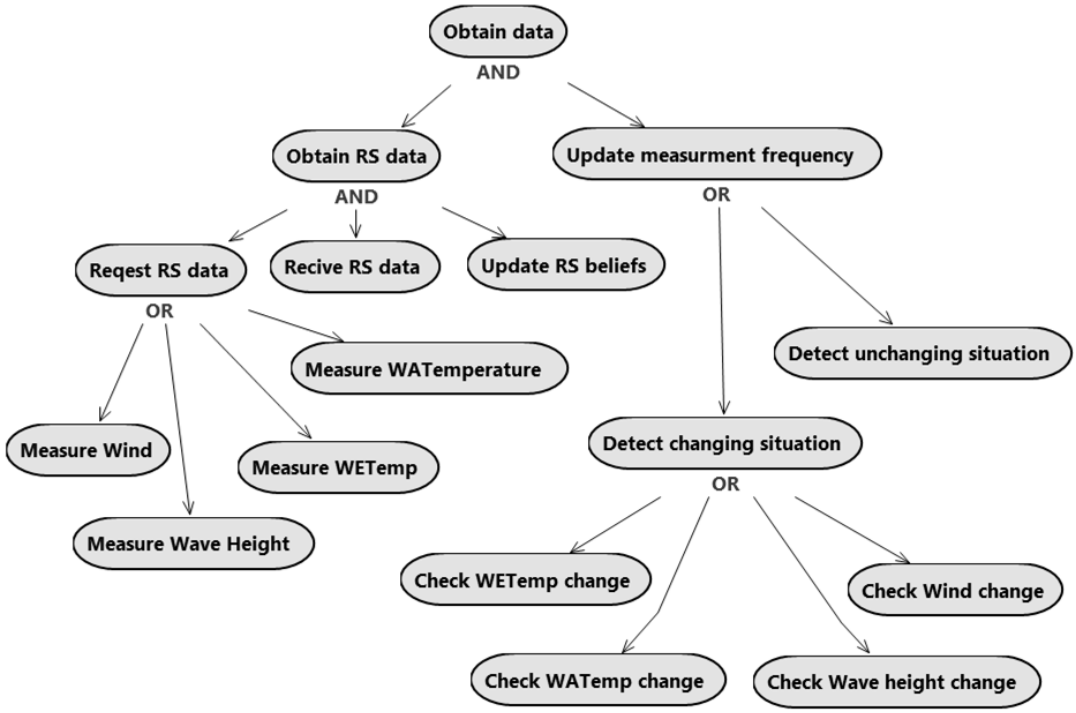
Veiksmai aprašo viską kas siunčiama iš sistemos į aplinką (arba kitas sistemas). Jų kuriamoje sistemoje identifikuoti trys:

„Request Data“ – plūdurą valdanti DAS gali pareikalauti techninės įrangos nuskaityti aplinkos parametrus;

„Sbscribe Buoy“ – veiksmas, kurio metu plūdurą valdanti DAS prisiregistruoja prie stebėjimų stoties;

„Log buoy data“ – nuskaitytų aplinkos parametrų perdavimas stebėjimų stočiai.

Tikslų diagrama (angl. *goal overview diagram*) yra aukščiausio lygio kurioje nurodomi visos sistemos ir atskirų komponentų tikslai, kuriuos sistemos komponentai turi įvykdyti. Šioje diagramoje kiekvienas tikslas gali būti sudarytas iš potikslių (žemesnio lygio tikslų), kurie gali būti sujungti „AND“ arba „OR“ valdymo elementais. Norint įgyvendinti tikslą, kurio potiksliai sujungti „AND“ elementais privalo būti įgyvendinti visi potiksliai. Įgyvendinti tikslą kurio potiksliai sujungti „OR“ elementais pakanka įgyvendinti bent vieną jo potikslį.



21 pav. *HDRS plūduro tikslų modelis*

Fig. 21. *Goals model of HDRS buoy*

Aukščiausio lygio tikslas HDRS sistemoje yra “Obtain data”, jis išskaldytas į du sub-tikslus:

„Obtain RS data“ – nuskaitomi realūs duomenys iš plūdure esančių jutiklių;

„Update measurement frequency“ – pagal aplinkos parametrų (banguotumo, temperatūros, vėjo kryptį ir greitį) kitimo greitį nustatomas ir atnaujinamas duomenų nuskaitymo ir persiuntimo stočiai dažnumas.

Tikslas „Obtain RS“ data yra išskaidytas į tris potikslius:

„Request RS data“ – reikalavimas nuskaityti duomenims iš plūduro jutiklių, šis tikslas savo ruožtu yra išskaldytas į keturis skirtingus tikslus: „Measure Wind“ (nuskaityti vėjo stiprumą ir greitį), „Measure Wave height“ (nuskaityti bangų aukštį), „Measure WETemp“ (nuskaityti oro temperatūrą), „Measure WATemperature“ (nuskaityti vandens temperatūrą). Kadangi vienu metu gali reikėti nuskaityti tik vieno tipo duomenis, todėl pakanka įvykdyti vieną iš šių tikslų tam, kad būtų įvykdytas „Request RS data“ tikslas.

„Receive RS data“ – gautų duomenų apdorojimas ir pavertimas į atitinkamą formatą.

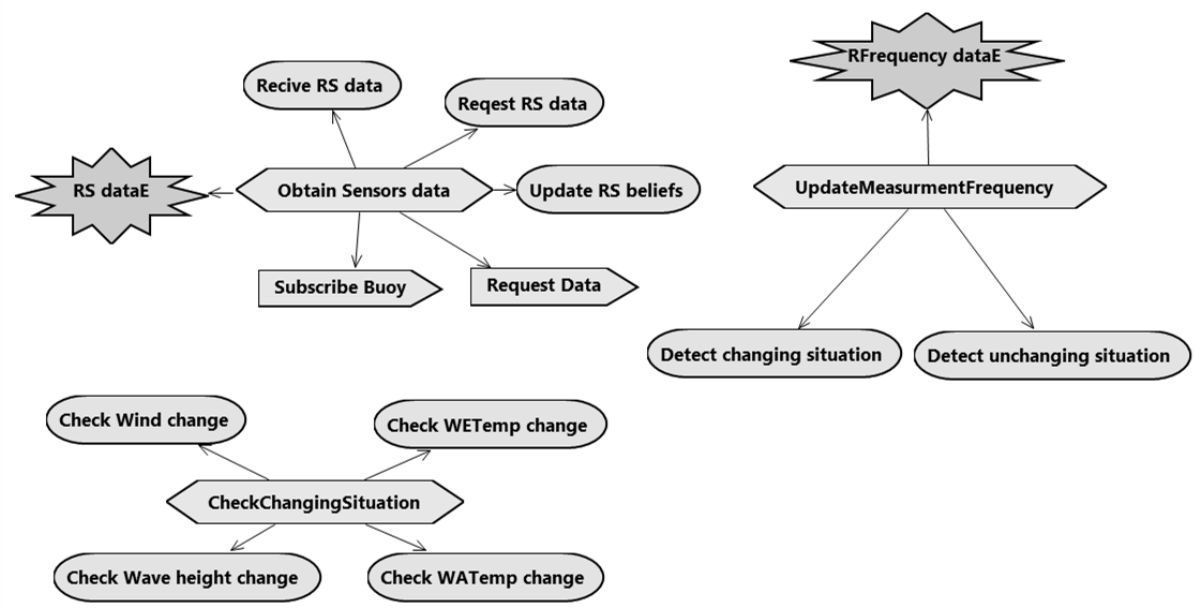
„Update RS beliefs“ – sistemai patalpinti gautus duomenis į atitinkamą duomenų saugyklą.

Tikslas „Update measurement frequency“ yra išskaldytas į du potikslius:

„Detect Unchanging situation“ – nustatomas nekintantys arba lėtai kintantys aplinkos parametrai (vykdomų nuoskaitų dažnumas sumažinamas);

„Detect changing situation“ – nustatomi greitai kintantys aplinkos parametrai (vykdomų nuoskaitų dažnumas padidinamas).

Sekantis sistemos projektavimo etapas sugrupuoti panašius tikslus į atskiras roles, kurios atvaizduojamos rolių diagrama (angl. *system roles diagram*). Perceptai ir veiksmai taip pat šioje stadijoje turi būti priskirti atitinkamoms rolėms. Šiame etape visi sistemos tikslai turi būti priskirti tam tikroms rolėms, tačiau jei tikslą sudaro keletas potikslių pakanka įtraukti į role pagrindinį tikslą, taip nurodant, jog potiksliai taip pat priklausys atitinkamai rolei.



22 pav. *HDRS plūduro valdymo rolių diagrama*

Fig. 22. *Roles model of HDRS buoy control*

HDRS sistemoje rolių diagramą išskaldyta į dvi dalis: rolės susijusios su plūduro valdymu (24 pav.) ir rolės susijusios su meteorologinės stoties (priimančios duomenis) valdymu (25 pav.). HDRS plūduro rolių diagramą sudaro 4 rolės:

„Obtain sensors data“ – rolė suteikianti galimybes gauti duomenis iš techninės įrangos, atlikti pirminį jų apdorojimą ir išsaugojimą;

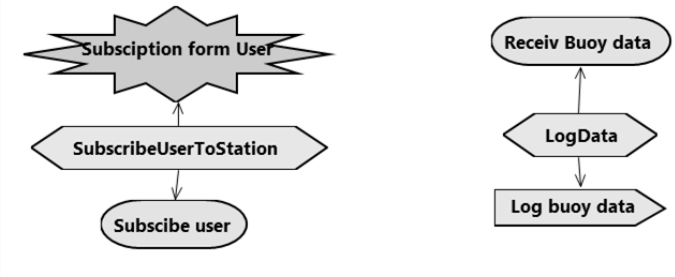
„UpdateMeasurmentFrequency“ – rolė suteikianti galimybes pakeisti nuoskaitų dažnumą ir pranešti apie pasikeitusią aplinkos situaciją netoliese esantiems plūdurams;

„CheckChangingSituation“ – rolė suteikianti galimybę nustatyti aplinkos parametrų kitimo greitį.

HDRS matavimų stoties rolių modelį sudaro dvi rolės:

„SubscribeUserToStation“ – užregistruojamas naujas matavimų stoties vartotojas;

„LogData“ – užregistruojami plūdurų atsiųsti duomenys.



23 pav. *HDRS matavimų stoties rolių modelis*

Fig. 26. *Roles model of HDRS measurements station*

Aprašius rolių modelį grįžtama prie bendrinės visos sistemos diagramos, kurioje kiekvienam sukurtam scenarijui priskiriamos jame naudotinos rolės ir įvykiai. Taip duomenų struktūros, kurios bus panaudotos tolimesniuose žingsniuose.

5.3. HDRS architektūros specifikavimas

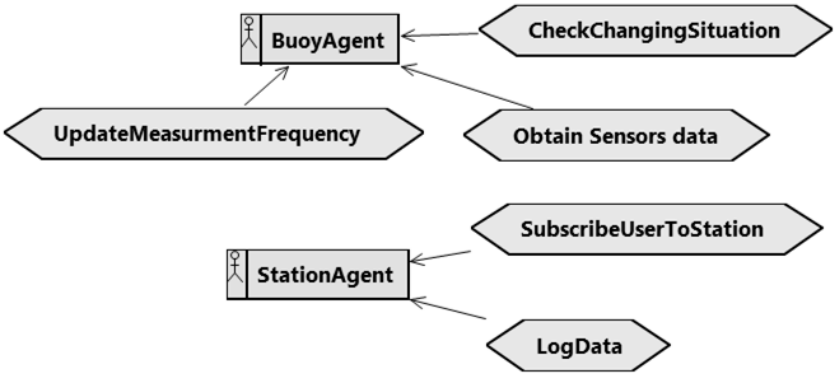
Sistemos architektūros specifikavimo etape atliekami trys pagrindiniai veiksmai: sugrupuojamos rolės į tam tikras grupes, kurios priskiriamos kuriamiems agentams, aprašomi agentų bendravimo protokolai (naudojant AUML notaciją) ir specifikuojamos duomenų saugyklos.

Agentų rolių išskyrimo etape būtina identifikuoti skirtingų tipų agentus ir jiems priskirti ankščiau specifikuotas roles. Sistemoje kiekvieno tipo agentų gali būti vienas arba keletas, diagramoje aprašomas kiekvienas tipas, o ne pats agentas. Išskiriant agentų tipus būtina atsižvelgti į kelis principus: jei įmanoma, rolės naudojančios tas pačias duomenų saugyklas, turėtų būti priskirtos to pačio tipo agentams; geriau kurta vieną agento tipą apjungiantį keleta rolių nei kiekvienai rolei kurta skirtingus agentų tipus.

Sukurtoje HDRS sistemoje galime išskirti dviejų tipų agentus kuriems priskiriamos ankščiau specifikuotos rolės (24 pav.):

„BuoyAgent“ – plūdurą aptarnaujantis agentas;

„StationAgent“- meteorologinių duomenų rinkimo stoties agentas.



24 pav. *Agentų rolių modelis*

Fig. 24. *Agents roles model*

Bendravimas tarp atskirų agentų ir aplinkos (kitų sistemų ar techninės įrangos) aprašomas protokolais ir specifikuojamas AUML diagramomis. Duomenys gali būti perduodami atskiromis žinutėmis arba aprašytais protokolais (protokolų pagalba atskiros žinutės sugrupuojamos į tam tikras bendravimo sekas). Kai bendravimui naudojama daugiau nei viena žinutė patartina šias žinutes apjungti į tam tikrą specifikuotą protokolą. Duomenų protokolų aprašui naudosime Michael Winikoff (Winikoff M., 2007) pasiūlyta AUML2 tekstinę notaciją, kuri vėliau gali būti atvaizduota grafiškai AUML diagramomis.

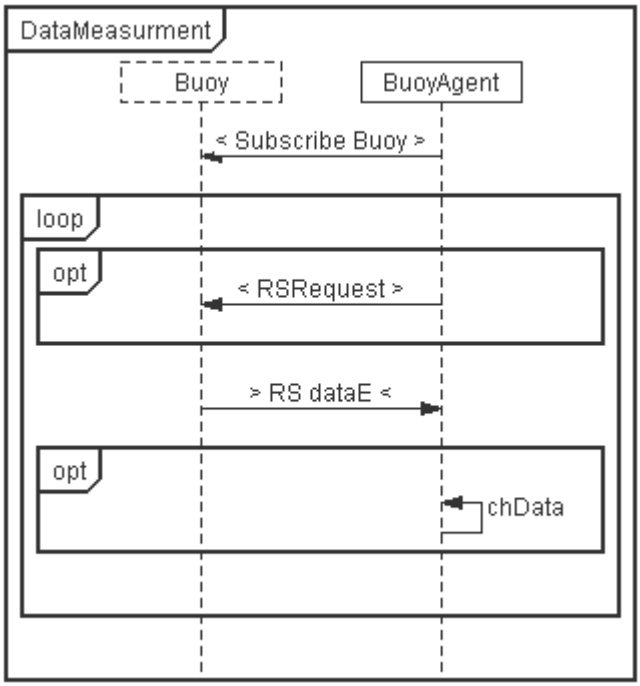
Hidrometeorologinių duomenų rinkimo sistemoje naudosime du duomenų perdavimo protokolus: matuojamų duomenų perdavimui iš plūduro techninės įrangos jį valdančiam agentui ir tų duomenų perdavimui netoli esantiems kitiems agentams ir kitą protokolą skirtą nuskaitytų duomenų perdavimui iš plūdurų į stebėjimo stotėį.

„DataMeasurment“ bendravimo protokolo specifikavimas pateikiamas žemiau:

```
start DataMeasurment
actor A Buoy
agent B BuoyAgent
  action B A Subscribe Buoy
  box loop
    box opt
      action B A RSRequest
    end opt
    percept A B RS dataE
    box opt
      message B B chData
    end opt
  end loop
finish
```

Protokolas apima dvi sistemos esybes: aktorių „Buoy“ (plūduro techninę įrangą) ir agentą „BuoyAgent“ (agentą kuris valdo plūdurą). Bendravimo protokolas suskaldytas į kelias dalis, pirmiausia agentas “BuoyAgent” įvykdo veiksmą “Subscribe Buoy”, kurio metu plūduro

programinė įranga „pririšama“ prie plūduro techninės įrangos (šio veiksmo metu nustatomi duomenų perdavimo protokolai, nustatoma įranga, kuri yra įrengta plūdure, inicializuojami I2C, SPI ir USART duomenų perdavimo protokolai, naudojami mikrovaldiklio ir sensorių bendravime). Sekanti protokolo dalis yra vykdoma ciklišk ai: pirmiausia agentas “BuoyAgent” gali pareikalauti duomenų nuskaitymo (šis veiksmas yra patalpintas į atskirą konteinerį “opt” ir pasirinktinai gali būti vykdomas arba nevykdomas kiekvienoje ciklo iteracijoje (agentas pats sprendžia kokiu dažnumu nuskaitomi duomenys iš jutiklių). Sekančiame žingsnyje jei duomenys yra siunčiami iš techninės įrangos agentas juos pasiima. Paskutiniu ciklo žingsniu jeigu nuskaityti duomenys skubiai keičiasi, agentas gali išsiųsti pasikeitusius duomenis kitiems, netoliese esantiems plūdurams, tam, kad būtų padidintas šių plūdurų duomenų nuskaitymo dažnis. Protokolo grafinis atvaizdavimas AUML diagrama pateikiamas 25 pav.



25 pav. *Matuojamų duomenų AUML diagrama*

Fig. 25. *Data collection AUML diagram*

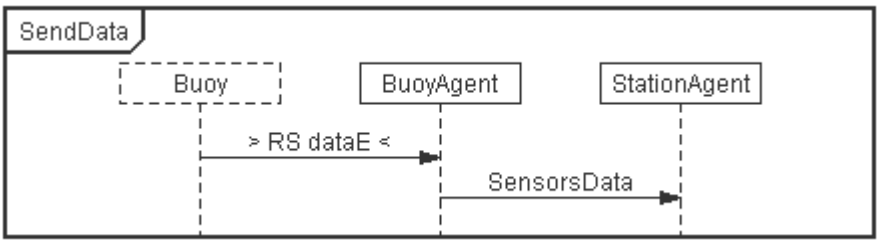
Kitas sistemoje naudojamas duomenų perdavimo protokolas skirtas išmatuotų duomenų perdavimui stebėjimų stoties agentui

```
start SendData
actor A Buoy
agent B BuoyAgent

agent D StationAgent
```

```
    percept A B RS dataE
    message B D SensorsData

finish
```



26 pav. *Išmatuotų duomenų siuntimo AUML diagrama*

Fig. 26. *Measured data transmission AUML diagram*

5.4. Hidrometeorologinių duomenų rinkimo sistemos techninė įranga

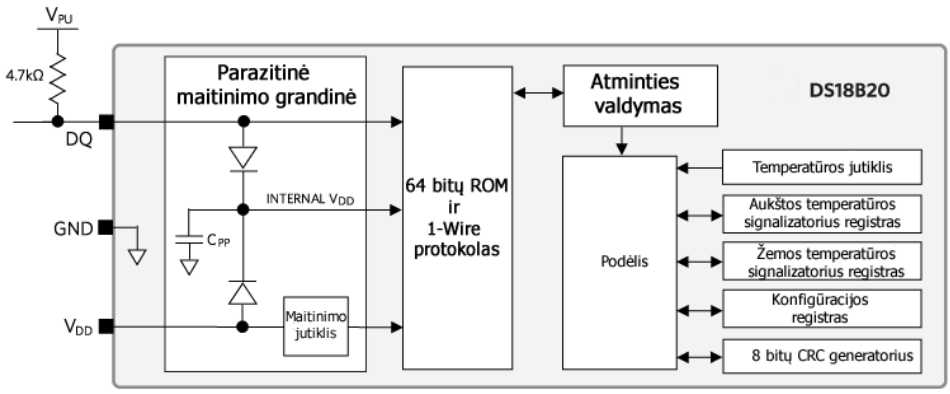
Naudojamos hidrometeorologinės sistemos kaupia informaciją apie šiuos aplinkos parametrus: vandens temperatūrą, vandens kolorimetrines savybes, rūgštingumą, oro temperatūrą, barometrinį oro slėgį, bangų aukštį, vėjo stiprumą, vėjo kryptį, kritulių kiekį ir k.t. Kuriamoje HMDRS sistemoje bus renkami šie parametrai vandens ir oro temperatūra, vėjo stiprumas ir kryptis, bangų aukštis ir kritulių kiekis.

5.4.1. Temperatūros duomenų rinkimas

Hidrometeorologinių duomenų rinkimo sistemos (HMDRS) kūrimo metu buvo atlikta įvairių temperatūros jutiklių analizė, jų tinkamumui jūros vandens ir oro temperatūrai matuoti. Jutiklių lyginamoji analizė buvo atliekama lyginant gamintojų pateikiamą informaciją apie jutiklių savybes. Analizei pasirinkti DS18B20, LM35, TMP36, ADT75 temperatūros jutikliai ir buvo lyginami šie parametrai: temperatūros matavimo ribos, tikslumas, vartojama elektros energija ir duomenų nuoskaitų greitis.

Atlikus temperatūros jutiklių analizę buvo nustatyta jog kuriamai sistemai tinkamiausias yra skaitmeninis temperatūros jutiklis DS18B20. Jutiklis gali matuoti temperatūrą nuo -55°C iki +125°C, termometro raiška 12 bitų, o paklaida 0.50°C (Maxim Integrated, 2008). Tačiau atlikus papildomus skaičiavimus ir matuojant temperatūros pokytį, šią paklaidą, galima sumažinti iki 0.10°C. Jutikliai yra gamykliškai sukalibruoti, todėl nereikalingas papildomas jutiklio kalibravimas prieš naudojimą. Jis gali būti maitinamas nuo

+3 iki +5.5 V įtampa. Budėjimo režime DS18B20 beveik nevartoja elektros energijos (srovė mažesnė nei 1 μA), o matavimų metu vartojama apie 1mA srovė, todėl jų panaudojimas HDRMS sumažins baterijų vartojimą, ir prailgins viso plūduro veikimo laiką be pakrovimo.



27 pav. Temperatūros jutiklio blokinė schema

***Fig. 27.** Temperature sensor block scheme*

Jutiklis vieną temperatūros matavimą atlieka greičiau nei per 0.7 sekundę. DS18B20 taip pat turi dar vieną privalumą prieš kitus analizuotus jutiklius – duomenų perdavimui naudojamas skaitmeninis 1-Wire® protokolas, kuris duomenims perduoti naudoja tik vieną kontaktą. Kiekvienas jutiklis turi unikalų 64 bitų numerį, kurio pagalba į vieną 1-Wire tinklą gali būti sujungti keleta jutiklių matuojančių skirtingas temperatūras.

5.4.2. Bangų aukščio matavimas

Jūros banguotumo matavimai gali būti atlikti skirtingais metodais, priklausomai nuo geografinės padėties, matavimų tikslumo ir kitų parametrų (Bykov, 2013). Pagrindiniai ir dažniausiai naudojami matavimo būdai:

- ultragarso jutikliais - naudojami matuoti bangoms kurių aukštis didesnis nei 5 metrai, jų pagrindinis trūkumas yra didelė matuojamų bangų aukščio paklaida;
- reostato tipo matuokliai - gali išmatuoti pakankamai tiksliai bangų aukštį, tačiau jiems reikalinga stacionari matavimų platforma;
- satelitinių nuotraukų analizė – naudojama tik ypač didelėms bangoms matuoti, taip pat negalimas realaus laiko matavimas;
- GPS sistemos – gali būti naudojamos išmatuoti bangoms kurių aukštis didesnis nei 2 metrai;

– akselerometras ir giroskopas – pasižymi nedidelė matavimo paklaida, nesudėtinga ir sąlyginai pigia įdiegimo kaina.

Pagal išskirtus privalumus ir trūkumus kuriamoje HMDRS bangų aukščio matavimui buvo pasirinkti naudoti akselerometras ir giroskopas. Bangų aukščio nustatymui (Panchang V., 1998) buvo panaudoti skaičiavimai pagal pateikiamas formules (1)-(3), kurių metu pirmiausiai pagal giroskopo pateikiamus duomenis, pakreipiamas giroskopą veikiančių jėgų parodymai statmenai žemės gravitacijai (1).

$$\begin{bmatrix} X_E \\ Y_E \\ Z_E \end{bmatrix} = \begin{bmatrix} a_1 & b_1 & c_1 \\ a_2 & b_2 & c_2 \\ a_3 & b_3 & c_3 \end{bmatrix} \begin{bmatrix} X_s \\ Y_s \\ Z_s \end{bmatrix} \quad (1)$$

Čia X_S, Y_S, Z_S yra pagreitis išmatuotas akselerometro pagalba, X_E, Y_E, Z_E yra pagreitis pakreiptas statmenai žemės gravitacijos jėgai.

Koeficientai a, b, ir c yra apskaičiuojami pagal sekančias formules:

$$a_1 = cos\theta cos\psi \quad (2)$$

$$b_1 = sin\varphi sin\theta cos\psi - cos\varphi sin\psi \quad (3)$$

$$c_1 = sin\varphi sin\theta cos\psi + cos\varphi sin\psi \quad (4)$$

$$a_2 = cos\theta sin\psi \quad (5)$$

$$b_2 = sin\varphi sin\theta cos\psi + cos\varphi sin\psi \quad (6)$$

$$c_2 = sin\varphi sin\theta cos\psi - cos\varphi sin\psi \quad (7)$$

$$a_3 = -sin\theta \quad (8)$$

$$b_3 = sin\varphi cos\theta \quad (9)$$

$$c_3 = cos\varphi cos\theta \quad (10)$$

Čia θ, ψ ir φ yra duomenys nuskaityti iš giroskopo.

Turint plūduro pagreičius - eliminuojama gravitacijos jėga veikianti plūdurą ir išreiškiamas pagreitis m/s^2.

$$A_x = -gX_E \quad (11)$$

$$\mathrm{A_y} = -gY_E \quad (12)$$

$$A_Z = g(1 - Z_e) \quad (13)$$

Čia Ax, Ay, and Az yra plūduro pagreitis pagal žemės koordinates, g – gravitacijos jėga.

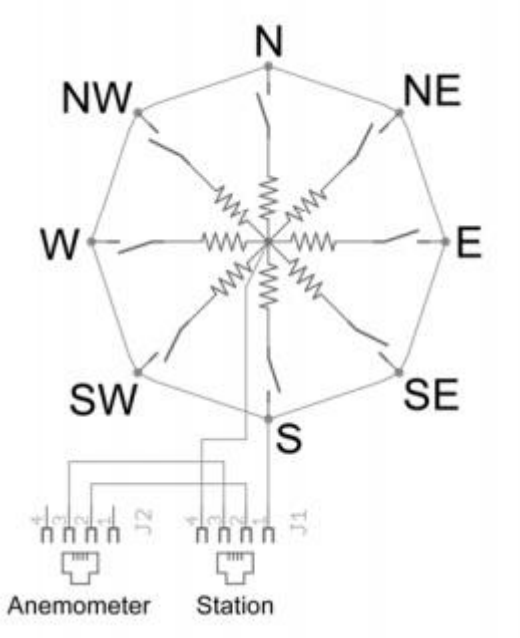
5.4.3. Meteorologinių duomenų matavimas

Sukurtoji HMDRS plūdurų sistema fiksuoja trijų tipų meteorologinius duomenis: kritulių kiekį, vėjo greitį bei vėjo kryptį.

Lietaus kritulių matuoklio veikimas yra pagrįstas, savarankiškai ištuštinama talpa į kurią surinkamas lietus, ir ištekant 0,2794 m^3 vandens iš talpos sujungiamas taktilinis jutiklis, t. y. veikia mygtuko principu, kuris gali būti registruojamas su skaitmeniniu mikrovaldiklio pertraukimu.

Anemometras (vėjo greičio matuoklis) fiksuoja vėjo greitį taktiliniu jutikliu uždarydamas jungiklį po kiekvieno apsisukimo, kuris įvyksta vėjui pasiekus 2,4 km/h arba 0,(6) m/s greitį.

Vėjarodės veikimo principas yra pagrįstas aštuoniais jungikliais, kurių kiekvienas jungiamas prie atskiro rezistoriaus (28 pav.). Vėjarodės magnetas gali uždaryti du jungiklius iš karto, todėl galima užfiksuoti iki šešiolikos skirtingų vėjo krypčių.



28 pav. Vėjarodės veikimo principas

***Fig. 28.** Weathervane operating principle*

Išorinis rezistoriaus yra naudojamas susidaryti įtampos daliklį, kurio pagalba yra išmatuojami vėjarodės parodymai analoginiu-skaitmeniniu keitikliu esančiu mikrovaldiklyje.

5.4.4. Komunikacijos protokolų lyginamoji analizė

Sekantis kuriamos plūdurų techninės įrangos žingsnis – pasirinkti duomenų perdavimo būdą. Kiekvienas iš jų turi tam tikrų privalumų ir trūkumų. Norėdami pasirinkti tinkamiausią reikia išskirti kiekvieno jų privalumus ir trūkumus. Skirtingi protokolai palyginti atsižvelgiant į reikalavimus, įskaitant duomenų perdavimo greitį, nuotolį, energijos sąnaudas. Taip pat svarbi palaikomo tinklo topologija. Suvestinė pateikiama palyginamojoje 5 lentelėje.

5 lentelė. Belaidžio tinklo duomenų perdavimo protokolų parametrų palyginimas

Parametrai	„Digimesh“	**Infrared**	**Wi-Fi**	**Bluetooth**
Panaudojimas	Monitoringas ir valdymas	Duomenų ir garso perdavimas	Internetas, namų tinklai	Kabelių jungčių pakaitalas
Tinklo topoligija	Ad-hoc, per-to-per, star, mesh	Ad-hoc	Point to hub	Ad-hoc, mesh
Darbo laikas su baterija, dienomis	100...1000+	1...2	1...5	1...7
Duomenų perdavimo greitis, bit	20...250 Kb/s	10 Mb/s	11...54 Mbit/s	720 Kb/s
Įrenginių sk. tinkle	65536	2	36	7
Veikimo nuotolis, m	1...1500	1...30	1...100	1...10+

Infraraudonųjų spindulių šviesa turi platų pralaidumą ir didelį greičio perdavimą (10 Mb/s), bet veikimo nuotolis yra sąlyginai mažas (iki 30 metrų). Galiausiai, infraraudonųjų spindulių technologija yra gana sena, todėl infraraudonųjų spindulių siųstuvai yra pigūs, palyginti su kitomis technologijomis. Infraraudonųjų spindulių transmisija taip pat turėtų būti stipri, kad netrukdytų kiti šviesos šaltiniai, pavyzdžiui, saulės spinduliai ir jų atspindžiai k.t. Kitas didelis šio perdavimo trūkumas yra tai, jog siųstuvas privalo tiesiogiai matyti imtuvą tam, tam kad duomenys būtų perduoti. Turbūt vienintelis šio duomenų perdavimo būdo privalumas yra tai, kad infraraudonųjų spindulių technologija yra gerai išvystyta, todėl infraraudonųjų spindulių siųstuvai-imtuvai yra pigūs, lyginant su kitomis technologijomis.

6 lentelė. IR ryšio panaudojimo galimybės

Privalumai	**Trūkumai**
Aukštas duomenų perdavimo greitis	Trumpo nuotolio veikimas
Integruojami pigūs komponentai	Pašaliniai šviesos trikdžiai

Wi-Fi technologija turi didžiausią duomenų perdavimo greitį ir sąlyginai didelį veikimo nuotolį iki 300 metrų (802.11b standartas). Duomenų perdavimo greitis priklauso nuo naudojamo standarto, 802.11b jis gali pasiekti iki 11 Mb/s. Tačiau WI-FI duomenų perdavimo technologija naudoja labai daug elektros energijos, todėl siųstuvo su baterijomis veikimo laikas labai trumpas.

7 lentelė. Wi-Fi ryšio panaudojimo galimybės

Privalumai	**Trūkumai**
Aukštas duomenų perdavimo greitis	Didelės elektros energijos sąnaudos
Didelis veikimo atstumas	Siųstuvas privalo tiesiogiai matyti imtuvą

„Bluetooth“ yra mažojo nuotolio duomenų perdavimo technologija (<10 metrų), tačiau šis intervalas gali lengvai būti padidintas, panaudojus galios stiprintuvą. Šis ryšio protokolas turi pakankamai mažas elektros energijos sąnaudas. Duomenų perdavimo greitis yra yra iki 720 Kb /s (antrosios versijos „Bluetooth“ standarte).

8 lentelė. „Bluetooth“ ryšio panaudojimo galimybės

Privalumai	**Trūkumai**
Aukštas duomenų perdavimo greitis	Mažas veikimo atstumas
Nedidelės elektros energijos sąnaudos	

„Digimesh“ protokolas skirtas naudoti įterptinėse sistemose, gali dirbti narvelio (angl. Mesh) tinklo topologija, palaiko 264 įrenginių viename tinkle. „Digimesh“ įrenginiai pasižymi nedideliu elektros energijos naudojimu. Duomenų perdavimo sparta yra palyginti mažia, ji siekia 250 Kb /s, tačiau tiek pilnai pakanka įrenginių valdymui ir duomenų nuskaitymui. „Digimesh“ gali aktyvuoti miegantį prietaisą (išeiti iš pristabdytosios veiksenos į aktyvią veikseną) per mažiau nei 15 milisekundžių, todėl didžiąją laiko dalį prietaisai gali būti energijos taupymo režime. „Digimesh“ įrenginiai vartoja labai mažai elektros energijos.

9 lentelė. „Digimesh” ryšio panaudojimo galimybės

Privalumai	**Trūkumai**
Didelis veikimo atstumas	Nedidelis duomenų perdavimo greitis
Labai mažos elektros energijos sąnaudos	

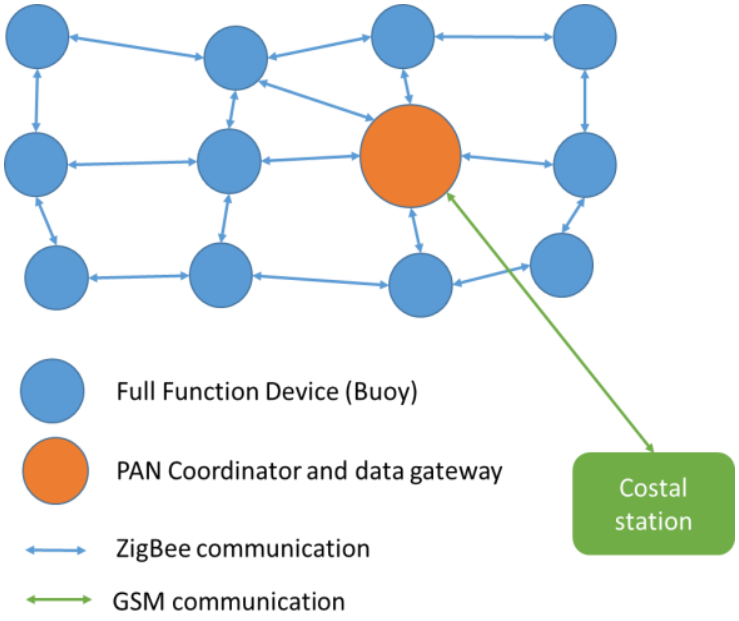
„Digimesh“ protokolas gali būti realizuotas XBee tipo moduliuose. Šiuos modulius naudosime mūsų kuriamoje sistemoje. Visi XBee moduliui duomenys perduodami UART protokolu, per kontaktų DIN eilę. Gauti duomenys surašomi į eilę ir tuomet perduodami radijo dažniu. Gauti duomenys yra išsiunčiami per DOUT kontaktus. Priimantis modulis duomenis perduoda UART protokolu jį valdančiam mikrovaldikliui. Moduliai gali būti konfigūruojami naudojant AT komandų režimą. Šią sąsają patogu naudoti prireikus valdyti ar perduoti duomenis vienam ar keliems identiškiems įrenginiams, kai negalima nurodyti gavėjo.

Pagal atliktą duomenų perdavimo būdų analizę, kuriamai HMDRS buvo pasirinkta naudoti „Digimesh“ duomenų perdavimo protokolas. Pagrindinės priežastys lemiančios šio protokolo pasirinkimą: mažos energijos sąnaudos, mesh tipo topologijos palaikymas, pakankamas duomenų perdavimo atstumas. „Digimesh“ yra atviras standartas, skirtas nedidelio nuotolio belaidžiams tinklams, pagrindinį dėmesį skiriant mažam energijos sunaudojimui ir dideliam tinkle patikimumui užtikrinti (Drungilas D., 2010).

„Digimesh“ protokolas naudoja trijų rūšių įrenginius:

1. Galiniai įrenginiai (angl. *end device*), kurie surenka duomenis ir juos išsiunčia.
2. Maršrutizatoriai (angl. *routers)*, kurie surenka duomenis iš galinių įrenginių ir juos perduoda kitam maršrutizatoriui arba gavėjui. Maršrutizatoriai taip pat gali veikti ir kaip galiniai įrenginiai (rinkti duomenis).
3. Koordinatorius (angl. *coordinator*), vienas iš tinkle maršrutizatorių tinkle turi būti nustatytas kaip koordinatorius, kuris nustato tinklo parametrus ir jį valdo, taip pat koordinatorius dažniausiai naudojamas kaip galinis įrenginys surinktų duomenų saugojimui.

Kuriamame HMDRS tinkle buvo panaudoti taip vadinamieji “visapusiškai veikiantys įrenginiai” (angl. *full function devices*), kurie surenka duomenis ir taip pat veikia kaip maršrutizatoriai duomenų perdavimui ir vienas koordinavimo įrenginys, kuris surinktus duomenis GSM tinklo pagalba perduoda pakrantės stočiai (29 pav.).



29 pav. MESH plūdurų tinklo struktūra

***Fig. 29.** Buoys MESH type network*

Pasiūlyta daugiaagentinė sensorinė sistema yra grindžiama tikslo siekimo principais, paskirstant užduotis agentams pagal jų gautų duomenų panašumą.

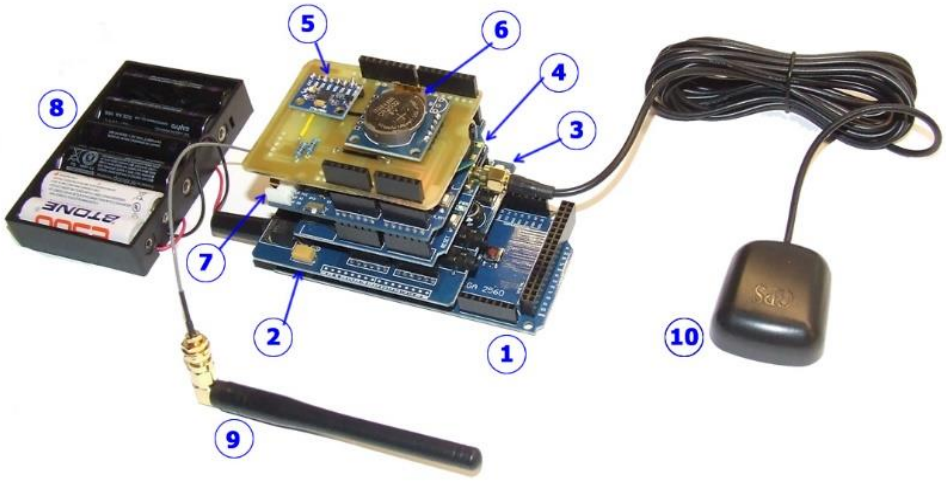
5.4.5. HDRS konstrukcinis realizavimas

Eksperimentiniam tyrimui, hidrometeorologinių stebėjimų plūdurams buvo sukurta modulinė elektronikos schema, kurios pagrindą sudaro (30 pav.):

1. Arduino Mega valdančioji plokštė su ATMega2560 mikrovaldikliu dirbančiu 16Mhz taktiniu dažniu.
2. SIM900 tinklo modulis leidžiantis persiųsti duomenis stebėjimų stočiai GSM tinklu.
3. GPS sistema kuri suteikia galimybę, nustatyti geografinę plūduro vietą.
4. 10mW XBee PRO modulis, kuris leidžia įgyvendinti bendravimą tarp atskirų plūdurų.
5. MPU6050 akselerometro/giroskopo pora skirta bangų aukščiui matuoti.
6. RTC laikmatis.
7. Jutiklių prijungimo schema skirta temperatūros, barometrinio slėgio ir kitų jutiklių prijungimui.
8. Maitinimo šaltinis.

9. GSM antena.
10. GPS antena.

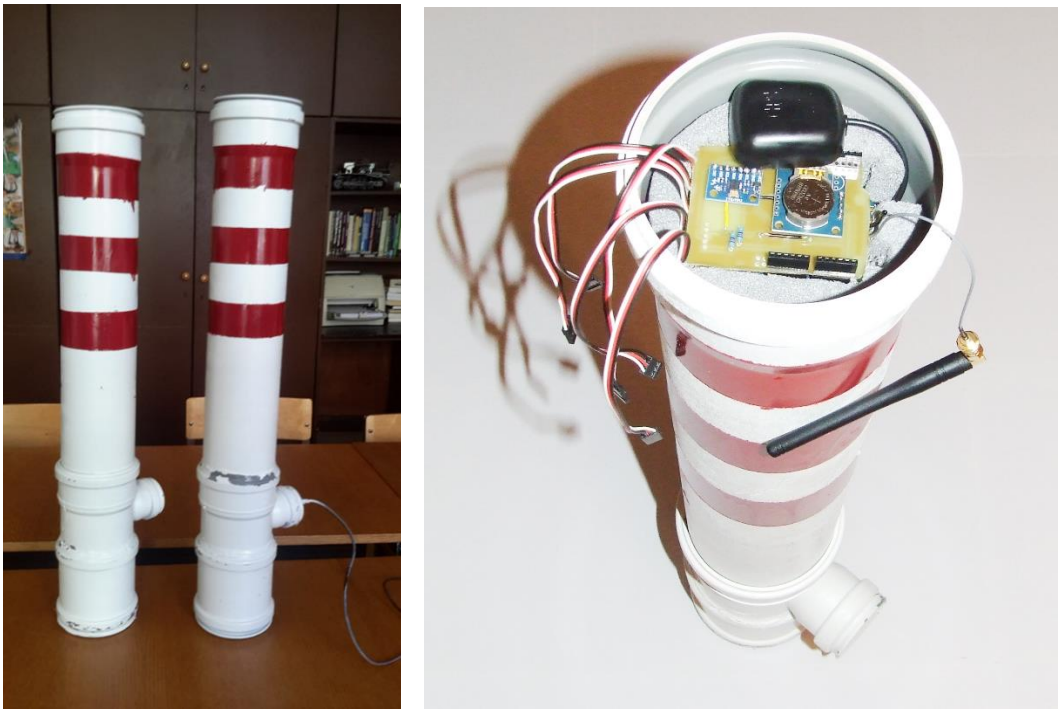
Kiekvienas plūduras gali būti maitinamas saulės energija, kuri tuo pačiu metu krauna Ni-Mh baterijas, tai leidžia plūdurui veikti tamsiu paros metu. Belaidžiam „Digimesh“ tinklui sukurti panaudoti 10mW XBee PRO moduliai, kurie duomenis gali perduoti 1-1.5 km. atstumu. Temperatūros matavimai atliekami DS18B20 jutikliai sujungtais į vieną 1-wire tinkle. Surinkti duomenys išsaugomi SD kortelėje ir perduodami į centrinę stotį per koordinatorių.



30 pav. Plūdurų elektronikos sudedamosios dalys

***Fig. 30.** Buoys electronic components*

Sukurtųjų plūdurų prototipų elektroninė dalis buvo patalpinta plastmasiniuose hermetiškuose korpusuose (31 pav.) ir išbandyti Baltijos Jūros priekrantėje. Eksperimentiškai išbandytas bendravimas tarp plūdurų panaudojant XBee PRO modulius buvo pasiektas ne mažesnis kaip 900 metrų duomenų perdavimo atstumas, tai yra pakankamas atstumas norint sudaryti pakankamo padengimo belaidį „MESH“ tinklą.



31 pav. Plūdurų konstrukcija

Fig. 31. *Buoys construction*

Meteorologiniams duomenims fiksuoti buvo panaudoti trys jutikliai (32 pav.): kritulių, anonimetras ir vėjarodė. Jų fiksavimui mikrovaldiklio ATMega2560 pagalba buvo panaudoti išoriniai valdiklio pertraukimai INT0 (kritulių kiekio fiksavimui) ir INT1 (vėjo greičio fiksavimui), bei analoginis skaitmeninis keitiklis vėjo krypčiai nustatyti.

32 pav. Meteorologinių duomenų rinkimui skirti jutikliai: lietaus kiekio, anemometras, vėjarodė

Fig. 32. *Meteorological data collection sensors:rain water amount, anemometer, windmeter*

5.5. HMDRS eksperimentinio tyrimo rezultatai

Duomenų rinkimo agentai, remiantis padarytais plūduro matavimais ir duomenimis gautais iš netoliese esančių plūdurų, nusistato prioritetus ir tai kaip dažnai agentas turi surinkti duomenis ir perduoti centrinei stočiai. Eksperimentinio tyrimo metu buvo lyginami dviejų tipų plūdurai: įprasti, duomenis centrinei stočiai perduodantys nustatytais laiko intervalais ir mokslinio tyrimo metu sukurtą daugiaagentinę sistemą naudojantys plūdurai, kurie duomenis nuskaito skirtingais dažniais (priklausomai nuo kintančios aplinkos parametrų).

Abiejų tipų plūdurams buvo nurodyta rinkti vienodo tipo duomenis: temperatūrą (vandens ir oro), bangų aukštį, vėjo greitį ir kryptį. Informacija perduodama duomenų paketais pateikiama 12 lentelėje.

10 lentelė. Surinktų duomenų paketo sandara

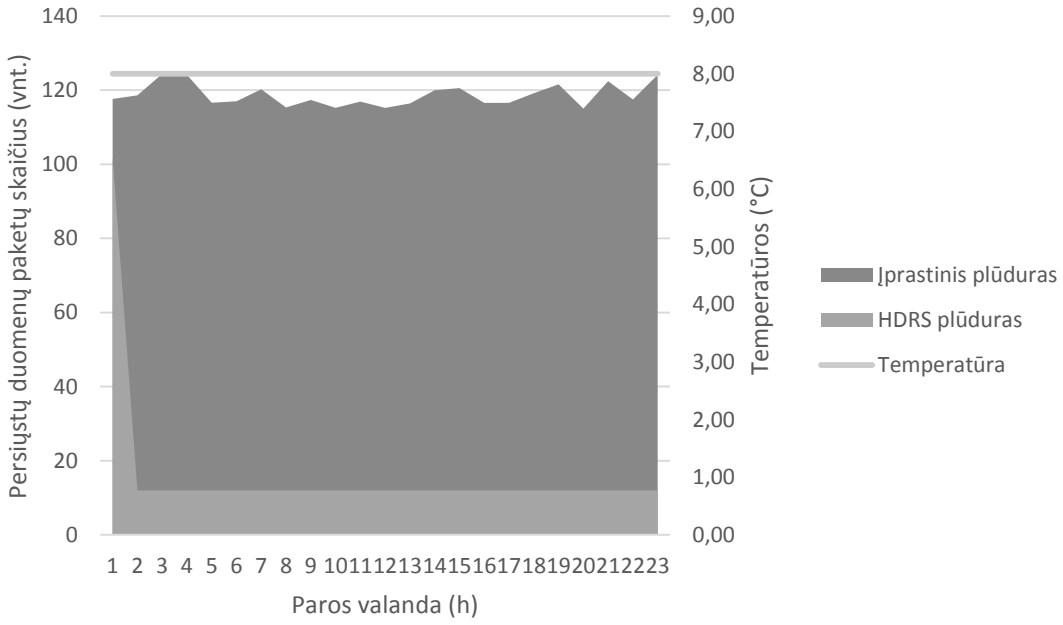
Perduodamos informacijos pavadinimas	**Užimama vieta**
Plūduro identifikacijos kodas	32 bitai
Matuojamo parametro identifikacijos kodas	8 bitai
GPS koordinačių ilguma	32 bitai
GPS koordinačių platuma	32 bitai
Vandens temperatūra	16 bitų
Oro temperatūra	16 bitų
Bangų aukštis	16 bitų
Vėjo greitis	8 bitai
Vėjo kryptis	8 bitai

Kiekvieną duomenų paketą sudaro plūduro identifikacijos kodas, plūduro geografinė padėtis, matuojamo parametro identifikacijos kodas ir perduodama matuojamo parametro reikšmė. Vidutiniškai 104 bitai informacijos (be papildomų DigiMesh antraščių). Šiuo metu naudojami plūdurai informaciją nuskaito ir perduoda visų matuojamų duomenų reikšmes centrinei stočiai vidutiniškai kas dvi – tris minutes. Tam, kad eksperimentų rezultatai būtų patikimi, visi eksperimentai buvo atliekami laboratorijos sąlygomis, abiejų tipų plūdurams siunčiant identiškas matuojamų parametrų reikšmes.

Pirmojo eksperimento metu visą matuojamąjį laiką t. y. 24 valandas plūdurams buvo siunčiamos pastovios (nekintančios) reikšmės į visus matuojamus jutiklius ir buvo matuojamas perduodamų paketų skaičius iš kiekvieno plūduro. 33 paveiksle pateikiami gauti rezultatai

pirmojo eksperimento metu, grafike atvaizduoti abiejų plūdurų perduodamų paketų skaičius per vieną valandą, o dešinėje ašyje atvaizduota į temperatūros jutiklį paduodama temperatūra (pastovi 8°C). Į visus kitus jutiklius visą laiką taip pat buvo paduodamos pastovios statinės reikšmės. Plūdurai nenaudojantys agentinės sistemos per vieną valandą vidutiniškai perdavė 119 duomenų paketų, tuo tarpu plūdurai su sukurtąja daugiaagentine sistema per pirmąją valandą perdavė 102 paketus, o vėlesniais matavimais tik po 12 duomenų paketų per valandą. Pirmąją valandą HDRS daugiaagentinė sistemai pradėjus fiksuoti duomenis bei atpažinus, jog matuojami parametrai nesikeičia sistema automatiškai pradėjo ilginti tarpus tarp atskirų nuskaitymų. Kai matuojamų parametrų reikšmės buvo vienodos HDRS visų matuojamų parametrų rinkinį (5 matuojami parametrai) centrinei stočiai perdavinėjo vidutiniškai kas 25 minutes.

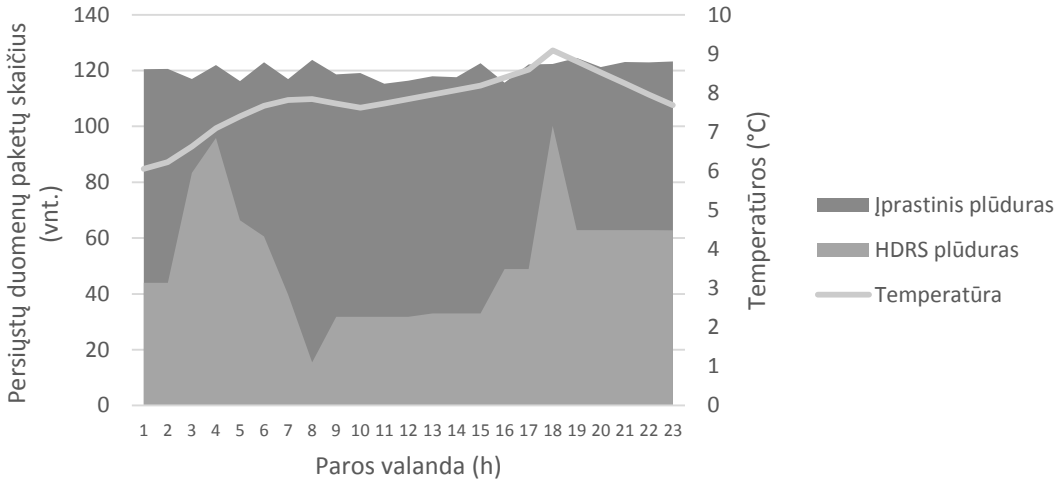
Šiuo eksperimentiniu tyrimu parodyta, jog naudojant pasiūlytąją daugiaagentinę sistemą plūdurams, esant nekintamoms aplinkoms sąlygoms, smarkiai sumažėja pasikartojančių duomenų perdavimo kiekis, lyginant su šiuo metu naudojamomis plūdurų sistemomis, kurios duomenis perduoda fiksuotais laiko intervalais. HDRS plūdurų sistemoje nuoskaitų skaičius 24 valandų intervale buvo 87% mažesnis nei naudojant fiksuotais laiko intervalais nuoskaitas darančių plūdurų.



33 pav. Persiunčiamų duomenų paketų kiekio kaita esant pastovioms klimatinėms sąlygoms

Fig. 33. *Transmissions of data packets under sustainable climatic conditions*

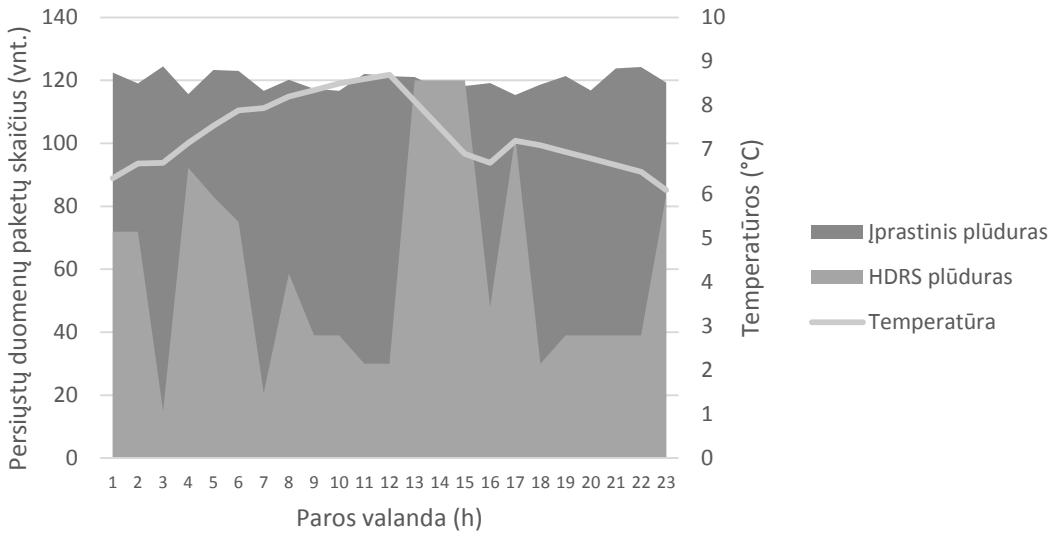
Antrojo eksperimento metu sukurtajai daugiaagentinei plūdurų sistemai buvo siunčiami realūs Baltijos jūroje išmatuoti aplinkos parametrai (34 pav.). Iš gautų rezultatų matyti, jog keičiantis aplinkos parametrams, priklausomai nuo greičio, daugiaagentinė sistema prisitaiko ir atitinkamai pagal aplinkos parametrų kitimo greitį reguliuoja duomenų nuskaitymo ir perdavimo dažnumą. HDRS plūdurų sistemoje nuoskaitų skaičius 24 valandų intervale buvo 63% mažesnis nei naudojant fiksuotais laiko intervalais nuoskaitas darančių plūdurų.



34 pav. Persiunčiamų duomenų paketų kiekio kaita esant lėtai besikeičiančioms klimatinėms sąlygoms

Fig. 34. *Transmissions of data packets under slowly changing climatic conditions*

Trečiojo eksperimento metu sukurtajai daugiaagentinei plūdurų sistemai buvo siunčiami realūs labai greitai besikeičiantys Baltijos jūroje išmatuoti aplinkos parametrai (35 pav.). Gautieji rezultatai parodė, jog sistema adaptuojasi prie ypač skubiai besikeičiančių parametrų smarkiai suintensyvindama duomenų nuskaitymą iš jutiklių.



35 pav. Persiunčiamų duomenų paketų kiekio kaita esant greitai besikeičiančioms klimatinėms sąlygoms

***Fig. 35.** Transmissions of data packets under rapidly changing climatic conditions*

Iš gautų rezultatų matyti, jog keičiantis aplinkos parametrams, priklausomai nuo greičio, daugiaagentinė sistema prisitaiko ir atitinkamai pagal aplinkos parametrų kitimo greitį reguliuoja duomenų nuskaitymo ir perdavimo dažnumą. Ties 13-16 valandomis buvo fiksuojamas staigus parametrų pasikeitimas, todėl šiame intervale agentinė sistema atliko po 121-122 nuoskaitas per vieną valandą, taip suteikdama galimybę detaliai ištirti skubiai besikeičiančius jūrinius procesus. HDRS plūdurų sistemoje nuoskaitų skaičius 24 valandų intervale buvo 46% mažesnis nei naudojant fiksuotais laiko intervalais nuoskaitas darančių plūdurų.

5.6. Penktojo skyriaus išvados

1. Penktajame skyriuje pateikiami hidrometeorologinių stebėjimų valdymo sistemos kūrimo eksperimentinių tyrimų metu gauti rezultatai. Tai nedidelio našumo įterptinių sistemų integravimo pavyzdys kurio sukūrimui buvo pasitelkta ankstesniuose skyriuose aprašyta DAS kūrimo metodika. Svarbi pasiūlytosios ir įgyvendintos daugiaagentinės sistemos savybė yra gebėjimas stebėti ir kaupti jūros hidrometeorologinius duomenis plačiu erdviniu padengimu bei pakankamai detalius laike palengvina jos praplečiamumą.
2. Praktiškai įgyvendintus hidrometeorologinių stebėjimų duomenų rinkimo sistemą sudarytą iš daugelio agentų buvo panaudota sukurtoji daugiaagentė sistema skirta

nedidelio našumo įterptinėms sistemoms. Eksperimentiniai rezultatai parodė, jog pasiūlytą sistema yra efektyvi taikyti nedidelio našumo įterptinėms sistemoms integruoti. Gauti eksperimentiniai rezultatai iliustruoja kaip sumažinama kompiuterinio tinklo apkrovimas, prie stebimų nekintančių (nuoskaitų skaičius sumažėjo 87%) arba lėtai kintančių aplinkos sąlygų ir kaip sistema adaptuojasi prie greitai besikeičiančių aplinkos sąlygų ir pateikia žymiai išsamesnius matavimus.

3. Svarbi pasiūlytosios ir įgyvendintos daugiaagentinės sistemos savybė yra gebėjimas stebėti ir kaupti jūros hidrometeorologinius duomenis plačiu erdviniu padengimu bei pakankamai detalius laike. Tokios daugiaagentinės sistemos naudojimas be prieš tai paminėtų privalumų, taip pat leidžia sumažinti elektros energijos sąnaudas (matavimai atliekami rečiau, todėl sensoriai ilgesnį laiko tarpą būna išjungti, sumažėja perduodamų duomenų dažnis, todėl siųstuvai gali būti ilgesnį laiką pervesti į miego režimą) ir palengvina jos praplečiamumą.

BENDROSIOS IŠVADOS

1. Išanalizavus metodikas, skirtas bendrosios paskirties daugiaagentinėms sistemoms kurti, nustatyta, kad jos yra per daug abstrakčios, jose neatsižvelgiama į nedidelio našumo įterptinėms sistemoms būdingas savybes ir apribojimus. Išnagrinėjus šių metodikų specializuotus variantus, skirtus įterptinėms daugiaagentinėms sistemoms kurti, nustatyta, kad jos taip pat negali būti tiesiogiai taikomos dėl šių priežasčių:
   - IĮS metodika grindžiama JADE karkasu, kurio vykdymui reikalinga papildomų resursų reikalaujanti JVM;
   - nei IĮS, nei DIAMOND metodika nesiūlo jas realizuojančios platformos ir kodo generavimo įrankių, skirtų nedidelio našumo įterptinėms sistemoms;
   - išdėstymo stadija (apimanti programinės/aparatinės įrangos atskyrimas) atliekama kūrimo ciklo pradžioje, todėl nėra galimybės integruoti aparatinę ir programinę įrangą viso daugiaagentinės sistemos kūrimo metu.
2. Tiriant DAS realizavimo platformas, nustatyta, kad dauguma jų yra skirtos didelį pajėgumą ir duomenų saugyklas turinčioms kompiuterinėms sistemoms veikiančioms prie stacionarių sąlygų. Nustatyta, kad autoriaus iškeltus reikalavimus iš dalies tenkina PDT „PROMETHEUS Design tool“ platforma, kuri leidžia generuoti pirminį programinį kodą įterptinėms sistemoms (iki klasių diagramų), bei gali būti integruojama su DIAMOND kūrimo metodika.
3. Atlikus eksperimentinius realaus laiko operacinių sistemų galimybių ir tinkamumo nedidelio našumo įterptinėms sistemoms tyrimą galima teigti jog tinkamiausia DAS kūrimui yra FreeRTOS operacinė sistema.
4. Modifikavus atrinktą kūrimo metodiką (DIAMOND) buvo pasiūlyta daugelio agentų sąveikavimu grindžiamos sistemos išplėtota infrastruktūra ir jos kūrimo metodika nedidelio našumo įterptinėms sistemoms kurti. Išdėstymo stadija, apimanti aparatinės ir programinės įrangos padalijimą, perkelta į kūrimo ciklo pabaigą, taip sudaro galimybę jungti aparatinės įrangos dalies ir programinės įrangos dalies vystymą viso daugiaagentinės sistemos kūrimo ciklo metu. Sudarytas transformavimo modelis skirtas, suprojektuotų DAS agentų atvaizdavimui į RTOS valdomą sistemą. Taip pat pateikiamas skirtingų tipų agentų realizavimo metodai nedidelio našumo įterptinėse sistemose.

5. Pasiūlyta DAS kūrimo metodika eksperimentiškai išbandyta sukuriant ir išbandant hidrometeorologinių duomenų rinkimo ir stebėjimo sistemą. Tokio tipo sistema taikoma sprendžiant ekologinio monitoringo uždavinius, esant realioms Baltijos jūros aplinkos ir klimatinėms sąlygoms. Gauti eksperimentiniai rezultatai demonstruoja, kai stebimi klimato ir aplinkos parametrai nekinta (arba kinta lėtai), sumažėja perduodamų nuoskaitų skaičius sumažėja 87%. Sistema demonstruoja tinkamą adaptyvumo prie greitai besikeičiančių aplinkos sąlygų savybę ir leidžia surinkti išsamesnius matavimų duomenis, nei įprastinės fiksuoto laiko sistemos.
6. Praktinio realizavimo metu buvo įvertinti duomenų perdavimo protokolai ir buvo nustatyta kad nedidelio našumo įterptinių sistemų integravimui į bendrą belaidžio tinklo sistemą – tinkamiausias DigiMesh belaidžių tinklų protokolas.

Literatūros sąrašas

Abidar, R., Moummadi, K., Medromi, H. 2011. Mobile device and multi agent systems: An implemented platform of real time data communication and synchronization. *Multimedia Computing and Systems (ICMCS)*, pp.1,6, 7-9

Allan R.J. 2009. Survey of Agent Based Modelling and Simulation Tools. Prieiga per Internetą: http://epubs.cclrc.ac.uk/bitstream/3637/ABMS.pdf

Almeida L., Pedreiras P. 2004. Scheduling within temporal partitions: response-time analysis and server design. *ACM Intl. Conference on Embedded Software(EMSOFT'04)*. pp.95 -103

Anand S. R., Georgeff M. P. 1999. BDI agents: From theory to practice. *Proceedings of the First International Conference on Multiagent Systems*.

Ancona, D., Demergasso, D., Mascardi, V. 2005. A Survey on Languages for Programming BDI-style Agents, *PROMAS Technical Forum.* Ljubljana, Slovenia.

Aylett R.S., Coddington A.M., Ghanea-Hercock R.A., Barnes D.P. 1995. Heterogeneous agents for multi-robot cooperation. *Design and Development of Autonomous Agents*, pp.3-7

Barrera C., Rueda M.J., Elgue J.C., Llinas O. 2006. Red ACOMAR: coastal moored buoy network for real-time surveillance, control and observation in Canary Islands. *OCEANS 2006*. pp.1–5.

Bauer B., Muller J., and Odeli J. 2000. Agent UML: A Formalism for Specifying Multiagent Interaction. *Agent-Oriented Software Engineering: First International Workshop, AOSE 2000.* pp. 91—103.

Bellifemine F., Caire G., Greenwood D. 1999. Developing multi-agent systems with JADE. *Wiley Series in Agent Technology.* ISBN 978-0-470-05747-6.

Bender L., Guinasso N.L., Walpert J.N., Howden S.D. 2010 A comparison of methods for determining significant wave heights. *Journal of Atmospheric and Oceanic Technology*,vol.27, no. 6, pp. 1012–1028,

Bordini R. H., Dastani M., Dix J., El Fallah Seghrouchni A., eds., Multi-Agent Programming: Languages. *Platforms and Applications.* Springer-Verlag. chapter 1, pp. 3-37

Bordini, R. H., Hübner, J. F., and Vieira, R. 2005. JASON and the Golden Fleece of agent-oriented programming. *Multi-Agent Programming. Multiagent Systems, Artificial Societies, and Simulated Organizations* Volume 15, pp 3-37

Bordini, R. H., Hübner, J. F., and Wooldridge, M. 2007. Programming Multi-Agent Systems in AgentSpeak Using Jason. John Wiley & Sons, Ltd. ISBN 978-0-470-02900-8

Bratman M. 1987. Intention, Plans, and Practical Reason. *Center for the Study of Language and Information*, ISBN 0674458184

Braubach L., Pokahr A., Lamersdorf W. 2005. Jadex: A BDI-Agent System Combining Middleware and Reasoning. *Software Agent-Based Applications, Platforms and Development Kits*, pp 143-168., ISBN 978-3-7643-7347-4.

Bykov D., Drungilas D., Andziulis A., Venskus J. 2013 Jūrų tyrimų monitoringo sensorinęs automatizuotos informacinės sistemos, skirtos išankstiniam ekologinių problemomų identifikavimui, projektavimo koncepcija. *Jūros ir krantų tyrimai. 7-oji nacionalinė Jūros mokslų ir Technologijų konferencija*, pp. 39-42

Carrasco A., Romero-Ternero M.C., Sivianes F., Hernández M.D., Escudero J.I. 2010. Multi-Agent and Embedded System Technologies Applied to Improve the Management of Power Systems. *International Journal of Digital Content Technology and its Applications* Volume 4, Number 1

Chella C., Cossentino M., Sabatucci L., Seidita V,. 2006 Agile PASSI: An Agile Process for Designing Agents. *International Journal of Computer Systems Science & Engineering. Special issue on "Software Engineering for Multi-Agent Systems"*, 21(2)

Chen W., GuanYu C.,; Liang H., 2011, Design of multi-interface Wi-Fi node for Sensor Network, *Artificial Intelligence, Management Science and Electronic Commerce (AIMSEC), 2nd International Conference* on , vol., no., pp.3902-3905

Collins C. 2012 In Situ Wave Measurements: Sensor Comparison and Data Analysis, *Open Access Theses*, University of Miami

Cossentino M. 2005. From Requirements to Code with the PASSI Methodology. *Agent-Oriented Methodologies*, Hershey, PA, USA.

D'Inve M. 2004. The dMARS Architecture: A Specification of the Distributed Multi-Agent Reasoning System. *Autonomous Agents and Multi-Agent Sytems*, No. 9, pp. 5–53

Dalamagkidis K., Kolokotsa D., 2008. Reinforcement Learning for Building Environmental Control. *Reinforcement Learning: Theory and Applications*, Book edited by Cornelius Weber, ISBN 978-3-902613-14-1, I-Tech Education and Publishing, Vienna, Austria, pp. 424.

Davis R. I., Burns A. 2005. Hierarchical fixed priority pre-emptive scheduling. *IEEE Real-Time Systems Symposium (RTSS'05)*, pp.389 -398

DeLoach S. A. 1999. Multiagent Systems Engineering: A Methodology and Language for Designing Agent Systems. *Agent-Oriented Information Systems (AOIS) '99*

DeLoach S. A. 2004. The MaSE Methodology. In Methodologies and Software Engineering for Agent Systems. *The Agent-Oriented Software Engineering Handbook Series Multiagent Systems, Artificial Societies, and Simulated Organizations*.

DeLoach S.A., Wood M.F., Sparkman C.H. 2001. Multiagent systems engineering. *International Journal of Software Engineering and Knowledge Engineering,* Vol. 11, No. 3 (2001) pp. 231-258.

Demazeau Y. 1995 From interactions to collective behavior in agent-based systems. *European Conference on Cognitive Science*, France

Demesure G. Defoort M., Bekrar A., Trentesaux D., Djemai M. 2014. Cooperation mechanisms in multi-agent robotic systems and their use in distributed manufacturing control: Issues and literature review. *Industrial Electronics Society, IECON 2014 - 40th* Annual Conference of the IEEE , pp.2538-2543

Drungilas D., Bielskis A. A., Denisov V. 2010. An intelligent control system based on non-invasive man machine interaction. *Innovations in Computing Sciences and Software Engineering*, pp. 63–68

Dzemydienė D. 2013. Sprendimų paramos sistemos galimybės vertinti vandens taršos procesus. *Jūros ir krantų tyrimai. 7-oji nacionalinė Jūros mokslų ir Technologijų konferencija*, pp. 69-72

Eswaran, A.; Rowe, A.; Rajkumar, R., 2005. Nano-RK: an energy-aware resource-centric RTOS for sensor networks, *Real-Time Systems Symposium, 2005* pp.,265

Finin T., Labrou. Y., 1997. KQML as an agent communication language. . J.M. In: Bradshaw (ed.), *Software Agents*, pp. 291-316. Cambridge, MA

Foundation for Intelligent Physical Agents. FIPA Web site. Prieiga per Internetą: http://www.fipa.org

Guo-quan W., Hai-Bin Y. 2004. Multi-agent reinforcement learning: an approach based on agents' cooperation for a common goal. *Computer Supported Cooperative Work in Design*, Vol.1, pp. 336-339.

Hagras, H.; Callaghan, V.; Colley, M.; Clarke, G.; Pounds-Cornish, A.; Duman, H. 2004. Creating an ambient-intelligence environment using embedded agents. *Intelligent Systems, IEEE* , vol.19, no.6, pp.12 - 20.

Inam, R.; Maki-Turja, J.; Sjodin, M.; Ashjaei, S.M.H.; Afshar, S., 2011. Support for hierarchical scheduling in FreeRTOS. *Emerging Technologies & Factory Automation (ETFA)*, pp.1-10

International Council for the Exploration of the Sea. Baltic Sea monitoring data. Prieiga per Internetą: http://ocean.ices.dk/Helcom/Helcom.aspx

JADE (JAVA Agent Development Framework). Prieiga per Internetą: http://jade.tilab.com

Jamont J. P. , Occello M., Lagreze A., A multiagent approach to manage communication. *Wireless Instrumentation Systems, Measurement*., Volume 43, Issue 4, ISSN 0263-2241, Elsevier, pp. 489-503, 2010.

Jamont J., Occello M. 2007. Designing Embedded Collective Systems: The DIAMOND Multiagent Method. Tools with Artificial Intelligence. *ICTAI 2007. 19th IEEE International Conference* pp.91-94

Jamont J.-P., Occello M. 2006. A self-organized energetic constraints based approach for modelling communication in wireless systems. *In Advances in Applied Artificial Intelligence*, volume LNAI N4031, pp. 101-110

Jamont J.-P., Occello M., Lagreze A. 2002 A multiagent system for the instrumentation of an underground hydrographic system. *In Proceedings of IEEE International Symposium on Virtual and Intelligent Measurement Systems*, Mt Alyeska Resort, USA

Jean-Paul J., Michel O. A., 2008 Multi agent Method to Design Open Embedded Complex Systems. *Tools in Artificial Intelligence*. ISBN: 978-953-7619-03-9

Jean-Paul Jamont, Michel Occello 2013, Using MASH in the Context of the Design of Embedded Multiagent System. *Advances on Practical Applications of Agents and Multi-Agent Systems*, ISBN 978-3-642-38072-3 , pp. 283-286

Juodelė B., Mikuckas A., 2007, Įterptinės sistemos didina produktų konkurencingumą. *Mokslas ir technika.* Vilnius

Kamal R. 2009. Embadded Systems, Architecture programming and Design. Tata McGraw-Hill Publishing Company Limited

Kasanen, E., Lukka, K. & Siitonen, A. 1993. The Constructive Approach in Management Accounting Research, *Journal of Management Accounting Research*, Vol.5, pp.241-264.

Khojasteh M.R., Meybodi M.R. 2005. Using Learning Automata in Cooperation among Agents in a Team. *Portuguese conference on Artificial intelligence 2005*, pp.306-312

Laarhuis J. H. 2010 Maritime Manet: mobile ad-hoc networking at sea. *Proceedings of the International Waterside Security Conference (WSS '10)*, pp.1–6

Li T. 2008 Multi-sink opportunistic routing protocol for underwater mesh network. *Proceedings of the International Conference on Communications, Circuits and Systems (ICCCAS '08)*, pp.405–409

Lin F. C., Yung-Jen Hsu, J. 1995. A decentralized cooperation protocol for autonomous robotic agents. Autonomous Decentralized Systems. *Proceedings of ISADS 95*. pp.420-426

Lind. J. 2001. Interative Software Engineering for multiagent systems:The MASSIVE Method, *LNCS/LNAI volume 1994*

Londhe S. N. 2008. Development of wave buoy network using soft computing techniques. *OCEANS 2008*—MTS/IEEE Kobe Techno- Ocean, pp.1–8

Marcus J. Huber M., Kumar S., Lisse S., McGee D. 2007 Integrating Authority, Deontics, and Deontics and Communications within a Joint Intention Framework, *Proceedings of the 2007 International Conference on Autonomous Agents and Multiagent Systems*

Maxim Integrated. 2008. DS18B20, Datasheat

McCabe F. G., Clark. K. L. 1995. APRIL—Agent PRocess Interaction Language. *Proceedings of the workshop on agent theories, architectures, and languages on Intelligent agents (ECAI-94)* Springer-Verlag New York, Inc., New York, NY, USA, pp. 324-340

McKeough T. 2009. House calls for the 21st century. *NEXT FUTURIST*

Meng, Y. 2005. An agent-based reconfigurable system-on-chip architecture for real-time systems. *Embedded Software and Systems, 2005. Second International Conference*

Mirza M. A., Shakir M. Z. Slim-Alouini M., 2011. A GPS-free passive acoustic localization scheme for underwater wireless sensor networks. *Proceedings of the 8th IEEE International Conference on Mobile Ad-hoc and Sensor Systems (MASS '11),* pp. 879–884

Mohamadi, T., 2011. Real Time Operating System for AVR microcontrollers. *Design & Test Symposium (EWDTS),* pp.376,380

Moulin B., Chaibdraa B. 1996. An Overwiew of Distributėd Artifical Inteligence. *Foundations of Distributed Artifical Inteligence*, pp 3-47

Nwana H. S. 1996 Software Agents: An Overview. *Knowledge Engineering Review*, Vol. 11, No 3, pp.1-40

O'Brien, P. D. and Nicol, R. C. FIPA , Towards a Standard for Software Agents. *BT Technology Journal*, vol. 13, issue 3, pp. 51-59

Occello M. 1998. Designing organized agents for cooperation in a real time context. *Collective Robotics*. pp. 25-73

Ostaševičiūtė L. 2009. Agentinėmis technologijomis pagrįstas intelektinių įterptinių sistemų kūrimas. *Daktaro disertacija: technologijos mokslai, informatikos inžinerija*. pp. 1-113

Padgham, L. and Winikoff, M. 2002. Prometheus: A pragmatic methodology for engineering intelligent agents. *Proceedings of the OOPSLA 2002 Workshop on Agent-Oriented Methodologies.* pp. 97-108

Panchang V., Zhao L., Demirbilek Z. 1998. Estimation of extreme wave heights using GEOSAT measurements. *Ocean Engineering*, vol. 26, no. 3, pp. 205–225

Pokahr A., Braubach L. ir Lamersdorf W., Jadex: A BDI Reasoning Engine. *Multi-Agent Programming*, 2005, pp 149-174

Pranevičius H., Raudys Š., Rudžionis A., Rudžionis V., Ratkevičius K., Sakalauskaitė J. Makackas D. 2008. Agentinių sistemų modeliai, Kaunas

Rao A. S. 1996. AgentSpeak(L): BDI agents speak out in a logical computable language. *Proceedings of the Seventh Workshop on Modelling Autonomous Agents in a Multi-Agent World (MAAMAW'96)*, no. 1038 pp. 42–55

Rao, A. S., & Georgeff, M. P. 1991. Modeling rational agents within a BDI Architecture. *Proceedings of the 2nd International Conference on Principles of Knowledge Representation and Reasoning.* pp. 473-484

Russel S., Norvig P. 1995. Artificial Intelligence : a Modern Approach. Prantice-Hall, 1995.

Sha L., Abdelzaher T., K-E. rzn, A. Cervin, T. Baker, A. Burns, G. Buttazzo, M. Caccamo, J. Lehoczky, Mok A. K. 2004. Real Time Scheduling Theory: A Historical Perspective. *Real-Time Systems*, vol. 28, no. 3, pp.101 -155

Shoham Y. 1993. Agent-orient programming. *Artificial Intelligence*, No. 60, pp. 51-92

Shoham Y. 1997. An Overview of Agent-oriented Programming. *Agent-oriented programming.* pp. 271-290

Sieber A., Cocco M., Markert J., Wagner M. F., Bedini R., Dario P. 2008. Zigbee based buoy network platform for environmental monitoring and preservation: temperature profiling for better understanding of mucilage massive blooming. *Proceedings of the 6th Workshop on Intelligent Solutions in Embedded Systems (WISES '08),* pp.1–14

Tapia, D.I., Alonso, R.S., Rodriguez, S., de Paz, J.F.; González, A., Corchado, J.M. 2010 Embedding reactive hardware agents into heterogeneous sensor networks. *Information Fusion (FUSION),* pp.1,8,

Texas Instruments. 2013. LM35 Precision Centigrade Temperature Sensors. Datasheat.

Vasile C., Pavel A., Buiu C. 2011. Integrating human swarm interaction in a distributed robotic control system. *Automation Science and Engineering (CASE),* pp.743,748

Wayne Wolf. 2008. Computers as components. Published by Elselvier Inc. ISBN:978-0-12-374397-8

Winikoff M. 2007. Defining syntax and providing tool support for Agent UML using a textual notation. *International Journal Agent-Oriented Software Engineering,* Vol. 1, No. 2, pp.123-144, 2007

Wooldridge M., Jennings N., Kinny D. 2000. The Gaia Methodology for Agent-Oriented Analysis and Design. *Journal of Autonomous Agents and Multi-Agent Systems,* volume 3. pp. 285-312

Wooldridge M., Jennings N.R. 1995. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review.* Vol. 10:2 p. 115-152

Wooldridge, M. 2006. An Introduction to MultiAgent Systems, John Wilet & Sons, LTD

Wu X., Peihuang L., Dunbing T. 2009 A multi-agent controller on embedded system for complex mechatronics, *Control and Decision Conference*. pp.3013-3018

Zambonelli F., Jennings N., Wooldridge M. 2003. Developing multiagent systems: The Gaia methodology. *ACM Transactions on Software Engineering and Methodology*, 12(3), pp. 317-370.

A PRIEDAS. Autoriaus mokslinių publikacijų disertacijos tema sąrašas

Straipsniai recenzuojamuose mokslo leidiniuose

1. Gricius G., Drungilas D., Andziulis A., Dzemydiene D., Voznak M., Kurmis M., Jakovlev S. 2014, Advanced Approach of Multi-Agent Based Buoy Communication, *The Scientific World Journal*, vol. 2015, p. 1-6 (Mokslinėje publikacijoje pateikiama autoriaus kurta daugiaagentinė sistema ir hidrometeorologinių plūdurų fizinė realizacija)
2. Bielskis A.A., Gricius G., Drungilas D., Dzemydienė D., Guseinovienė E., Ambient Lighting Controller Based on Reinforcement Learning Components of Multi-Agents. *Electronics and Electrical Engineering*, ISSN 1392 – 1215, 5 (121), p. 79 – 84, Kaunas, Lithuania, 2012 (Mokslinėje publikacijoje pateikiama autoriaus projektuota ir programiškai realizuota daugiaagentinė sistema)
3. Gricius, G., Bielskis, A.A., Denisovas, V., Drungilas, D., Dzemydienė, D. Multi-Agent-Based human Computer Interaction of E-Health Care System for People with Movement Disabilities. *Electronics and Electrical Engineering.* ISSN 1392-1215, 7(103), p. 77-82, Kaunas, Lithuania, 2010 (Mokslinėje publikacijoje pateikiama autoriaus projektuota ir programiškai realizuota daugiaagentinė sistema)
4. Dzemydiene, D., Bielskis, A.A., Andziulis A., Drungilas, D., Gricius, G. Recognition of Human Emotions in Reasoning Algorithms of Wheelchair Type Robots. *Informatica*. Vol.21 Issue 4, p. 521-532. Vilnius, Lithuania, 2010 (Mokslinėje publikacijoje pateikiama autoriaus projektuota ir programiškai realizuota daugiaagentinė sistema)

Straipsniai kituose mokslo leidiniuose

1. Gricius G., Drungilas D., Guseinovaitė J., Grigaitis K., Bielskis A., Modeling of Human Friendly Multi-Agent Based Sustainable Power Controller. *Balkan journal of electrical & computer engineering.* ISSN 2147-284X, vol.1, no.2 , Kirklareli, Turkey, 2013 (Mokslinėje publikacijoje pateikiama autoriaus projektuota daugiaagentinė sistema, įgyvendinta techninė įranga)
2. Bielskis A., Drungilas D., Gricius G., Guseinoviene E., Dzemydiene D., Žutauta L., Modeling of Ambient Comfort Affect Reward Based on Multi-Agents in Cloud Interconnection Environment for Developing the Sustainable Home Controlle. *Proceedings of 2013 Eighth International Conference and Exhibition on Ecological*

Vehicles and Renewable Energies (EVER), ISBN: 978-1-4673-5270-3, EEE Catalog Number: CFP1328U-CDR, Monaco, 2013 (Mokslinėje publikacijoje pateikiama autoriaus realizuota techninė ir programinė įranga)

3. Gricius G. , Ramašauskas O., Non-rigid image recognition algorithms in applied robotics, *Proceedings of International Workshop STOPROG-2012, Stochastic Programming for Implementation and Advanced Applications*, ISBN 978-609-9524 1-4-6, *Neringa*, Lithuania, 2012 (Mokslinėje publikacijoje pateikiama autoriaus sukurta techninė įranga ir programinis algoritmų įgyvendinimas)

Gediminas Gricius

DAUGIAAGENTINIŲ SISTEMŲ KŪRIMO METODŲ IŠVYSTYMAS NEDIDELIO NAŠUMO ĮTERPTINIŲ SISTEMŲ INTEGRAVIMUI

Daktaro disertacija
Fiziniai mokslai,
Informatika (09 P)

Redaktorė Toma Arcišauskaitė-Lukošienė