Vilniaus universitetas
Duomenų mokslo ir skaitmeninių technologijų institutas
L I E T U V A

INFORMATIKA (N009)

INTELEKTUALIZAVIMO METODŲ IŠVYSTYMAS APLINKOS STEBĖJIMO IR VALDYMO SISTEMOJE, INTEGRUOJANČIOJE ĮTERPTINES KOMPONENTES

Vytautas Radzevičius

2019 m. spalis

Mokslinė ataskaita DMSTI-DS-N009-19-14

Disertacinio dabo vadovas: prof. Dr. Dalė Dzemydienė

VU Duomenų mokslo ir skaitmeninių technologijų institutas, Akademijos g. 4,
Vilnius LT-08412
www.mii.lt

Santrauka

Darbe yra nagrinėjami skirtingi būdai ir požiūriai kaip galima intelektualizuoti aplinkos stebėjimo sistema , kad ši galėtų veikti autonomiškai, interpretuoti duomenis iš skirtingų šaltinių ir jai reiktų kuo mažiau priežiūros. Darbe nagrinėjamos daugiaagentinės sistemos, DEVS modelis, neuroninių tinklų panaudojimas, taip pat visi šie modeliai yra komponuojami , kad būtų galima ne tik stebėti bet ir reaguoti į aplinkos pokyčius ir nustatyti pasikeitusias, ar gresiančias aplinkos būsenas.

Reikšminiai žodžiai: intelektualizavimo metodai, jūros vandens aplinkos stebėjimas, stebėjimo plūdurai, įterptinės sistemos.

TURINYS

ĮVADAS

Temos aktualumas. Kuriamos modernios įterptinės sistemos reikalauja intelektualizavimo metodų, grindžiamų tam tikrais dirbtinio intelekto metodais, tokiais kaip mašininis mokymasis, neuroniniai tinklai, sprendimų medžiai, daugelio agentų sistemos. Vis daugiau atsiranda įrenginius valdančių sistemų, stebinčių aplinkos parametrus, kurios prijungtos prie tinklo ir įgalina nustatyti ir identifikuoti aplinkos situacijas realiu laiku. Prognozavimo sistemos leidžia numatyti kaip gali pasikeisti aplinka po tam tikro laiko. Operatyvaus valdymo sistemos dažniausiai yra valdomos tiesioginių reikšmių stebėsenos ir situacijų nustatymų principais.

Intelektualizavimo metodai įgalina realizuoti daugelį valdymo uždavinių ir reikalingi sudėtingose sistemose. Dažniausiai tokiose sistemose intelektualizavimo metodai įgyvendinami sudėtingomis sąlygomis. Išmaniųjų paslaugų sistemos projektuojamos atsižvelgiant į daiktų interneto suteikiamas galimybes, kur reikalingi aplinkos stebėjimai ir jų analizės įverčiai, pasitelkiant mašininį mokymąsį kuris gali būti taikomas priėmimui (Geraldo P. R. Filho ir kt., 2018; Afshin Najafi-Ghalelou ir kt., 2018; K.S.Gayathri ir K.S.Easwarakumar., 2016).

Jutiklių tinklai ir įterptinės sistemos, sujungtos tarpusavyje vienam tikslui pasiekti yra pagrindinės daiktų interneto sistemų įgyvendinimo komponentės (De ir kt., 2012; De Silva ir kt., 2012; Murtaza ir kt., 2013). Vykdant sprendimų priėmimą, dažnai susiduriama su neapibrėžtomis situacijomis, kai reikia interpretuoti, skirtingus duomenis ir pagal juos nuspręsti koks veiksmas turėtų būti teisingiausias kiekvienu skirtingu atveju (Jonathan S.Lee ir kt., 2017).

Nepaisant didelės dirbtinio intelekto (DI) raidos, DI metodai nėra pakankamai taikomi operatyvaus valdymo sistemose, kuriose integruotos įterptinės sistemos (sensoriai, valdikliai, jutikliai ir pan.). Aktuali problema yra, kad operatyvaus valdymo sistemose visi veiksmai, turi būti priimami pakankamai greitai, o įterptinės sistemos turi mažus skaičiavimo galingumus, kurių nepakanka sistemos intelektualizavimui.

Dar viena aktuali problema yra - duomenų perdavimo protokolų suderinamumo tarp skirtingų sistemos dalių užtikrinimas. Tai kelia tam tikrus naujus reikalavimus bendram didelių srautų informacijos išsaugojimui, susisteminimui ir žinių išgavimui iš didelių duomenų saugyklų.

Kadangi aplinkos stebėsenai sistemos yra išskirstytos, dirbančios belaidžio tinklo sąlygomis, kai stebėsenai skirti įrenginiai susidūria su energijos tinkamo tiekimo problema. Tenka spręsti energijos tiekimo prailginimo periodų ir išskirstymo efektyvumo klausimus. DI metodai gali padėti prognozuoti ir numatyti energijos pasiskirstymo algoritmus, bei padidinti efektyvumą kuris ypač reikalingas įrenginiuose nutolusiai naudojančiuose energiją. Kai kuriuos energijos skirstymo klausimus autoriai nagrinėja darbuose (Mahmoud ir kt., 2013; Paramanathan ir kt., 2012; Veleva ir kt., 2012; Yuan ir kt., 2012).

Iš kitos pusės, nagrinėjant aplinkos stebėjimo klausimus, bus apsiribojama vandens stebėsena Baltijos jūros pakrantėje įtaisytais ir tam tikslui sumontuotais pludurais. Iškylančios problemos su susidariusių vandens pokyčių situacijų analizę pakankamai aktualūs jūrų vandens kokybei tirti ir analizuoti. Tai problemos susijusios ir su daugelio įtakos faktorių analizę. Labai svarbiais susijusiais klausimais tampa upių ekosistemų stebėjimas, dirvožemio užterštumas, lietaus vandens nuoplovos ir pan. Stebimos situacijos, kai padidėjus taršai Baltijos jūros vanduo žydi dažniau,

atsiranda papildomos invazinės rušys dėl balastinių vandenų. Į upes yra išleidžiamos nepilnai išvalytos nuotekos dėl kurių žūstą žuvys, mažėja žuvų populiacija ir visa tai daro įtaką finansiniams nuostaliams tiems regionams, kurie yra paliečiami vandens užterštumo. Vandens kokybės stebėjimas yra daugiakomponentinis. Viena, kai atliekamas momentinis patikrinimas. Tačiau vykdomi ir atliekami išsamūs meteorologiniai, biologiniai ir cheminiai stebėjimai, keliuose taškuose ir meteoroligių duomenų sekimas keliose stacionariose stotelėse, priekrantėje ir nutolusiuose mazguose.

Tačiau stebėjimui ir vykdomai kontrolei nepakanka pusiau automatizuotų sistemų. Norint pastebėti taršos atvejus, reikia padengti didelę geografinę teritoriją ir visą jutiklių tinklą sujungti į sinchronizuotai ir daniai veikiančią sistemą. Daugiakomponentinėje sistemoje iškyla poreikis panaudoti intelektualizavimo metodus, kad būtų galima tinkamai vykdyti operatyvų stebėsenos procesą ir teisingai identifikuoti padidėjusios taršos atvejus.

Akcentuojant operatyvaus jūrų vandens taršos nustatymo poreikį, išskiriame pagrindines problemas, kuias reikia spęsti informatikos mokslo kontekste.

Sprendžiamos problemos:

- Nepakankamas intelektualizavimo metodų taikymas operatyvaus jūrinio vandens būklės stebėjimo ir taršos automatinio nustatymo sistemose, reikalauja naujo integruoto požiūrio į šių metodų taikymą;
- Nepakankamai išplėtota infrastruktūra, kuri apimtų duomenis daugiau nei iš vieno nutolusio stebėsenos šaltinio ir gebėtų analizuoti stebėsenos duomenų dideles saugyklas, reikalauja atitinkamos belaidžių tinklų platformos išvystymo ir papildomų duomenų saugyklų analizės metodų, įgalinančių adekvatų situacijų atpažinimą;
- Nors jūros vandens taršos analizės metodų pasiūlyta literatūroje gana daug, tačiau jų taikymas operatyviai veikiančioms ir nedidelius pajėgumus turinčioms įterptinėms sistemoms yra apribotas, dėl ko tenka ieškoti naujų sprendimų ir būdų tinkamai spręsti jūros vandens situacijų identifikavimo uždavinius, integruojant keletą dirbtinio intelekto metodų, kurie galėtų veikti sistemoje kuri dirbtų išskirstyto belaidžio plūdurų tinklo sąlygomis, pasitelkiant įterptinių stebėsenos sistemų komponentes.

Tyrimų objektas

Metodai ir priemonės leidžiančios užtikrinti daugiakomponentinių įterptinių sistemų darbą ir situacijų analizę bei kritinių situacijų atpažinimą belaidžio tinklo komunikavimo infrastruktūroje.

Mokslinio **darbo objekto detalesnis aprašymas**

Darbe bus nagrinėjama metodika ir daugiakomponentės sistemos architektūros veikimo priemonės, kurių pagalba būtų galima sistemingai pritaikyti keletą dirbtinio intelekto (DI) metodų (tokių kaip mašininio mokymosi metodai, taisyklių sistemos, dalimis tiesiniai agregatai, daugelio agentų sąveikos mechanizmai ir scenarijai), nustatant pažeistinas situacijas jūros vandens ekvatorijuje, prognozuoti ne tik einamuosius sprendimus, bet ir bandyti spręsti apie eko-sistemos būsenas ateityje.

Disertacinio dabo tikslas:

Išplėtoti dagia-agentinių sistemų sąveikos infrastruktūrą, įgalinant tinkamą operatyvų jūros vandens stebėjimą ir pasiūlyti integruotus DI metodus sprendimų priėmimui.

Tikslui pasiekti keliami šie uždaviniai.

Disertacinio darbo uždaviniai

1. Išnagrinėti dirbtinio intelekto metodus, taikomus daugiaagentinių sistemų kūrimui, parenkant tinkamus jūros vandens monitoringui atlikti belaidžių tinklų pagrindu.
2. Išanalizuoti taikomus intelektualizavimo metodus, leidžiančius priimti sprendimus mažo našumo įterptinių posistemių integravimo į bendrą sistemą sąlygomis.
3. Išplėtoti infrastruktūrą sistemos, kuri apjungtų duomenis daugiau nei iš vieno nutolusio šaltinio ir gebėtų analizuoti stebėsenos duomenų dideles saugyklas
4. Pasiūlyti tinkamus intelektualizavimo metodus, kuriuos galima būtų įgyvendinti daugelio jutiklių tinklo integravimo aplinkos valdymo sistemoje, išvystant valdymo sprendimų priėmimo ir prognozavimo galimybes.

Tyrimų metodai

Sisteminė literatūros analizė, palyginamieji metodai. Daugiakriteriniai metodai ir priemonių architektūrų analizė, kuri atskleistų technines galimybes, leidžiančios užtikrinti daugiakomponentinių įterptinių sistemų darbą ir situacijų analizę bei kritinių situacijų atpažinimą.

1 INTELEKTUALIZAVIMO METODŲ POREIKIS IR GALIMYBĖS VANDENS TARŠOS STEBĖSENAI IR SITUACIJŲ ATPAŽINIMUI

Skyriuje bus apžvelgiami kai kurie daugiaagentinių sistemų intelektualizavimo metodai.

1.1 Intelektualizavimo metodų poreikis vandens taršos stebėsenai ir situacijų atpažinimui

Tinkami intelektualizavimo metodai labai reikalingi, kad galima būtų realizuoti daugelį valdymo uždavinių sudėtingose reliomis sąlygomis veikiančiose sistemose. Dažniausiai tokiose sistemose intelektualizavimo metodai įgyvendinami apribotomis veikimo sąlygomis. Kai kurių išmaniųjų paslaugų sistemos projektuojamos atsižvelgiant į daiktų interneto technologinių platformų suteikiamas galimybes, kur reikalingi aplinkos stebėjimai ir jų analizės įverčiai. Žinomi darbai, kuriuose pasitelkiami mašininio mokymosi metodai kurie gali būti taikomi sprendimų priėmimui (Geraldo P. R. Filho ir kt., 2018; Afshin Najafi-Ghalelou ir kt., 2018; K.S.Gayathri ir K.S.Easwarakumar., 2016).

Baltijos jūros pakrantėje esančios priekrantės, bei upės ir jų baseinus sudarančios kiti šaltiniai nėra stebimi pastoviai, yra atliekami tik momentiniai kokybės patikrinimai, bei esančios pastovios aplinkos stebėjimo kurios matuoja vėjo greitį, vėjo kryptį, ištirpusio deguonies kiekį vandenyje, temperatūrų skirtinguose gyliuose. Visos šios stacionarios sistemos atlieka tik kaupiamąjį vaidmenį, kur yra tiesiog surenkami duomenys ir niekur toliau nenaudoji nebent retrospektyviam duomenų apdorojimui po ilgo laiko. Dėl to mes nagrinėjame priemones kuriomis būtų galima pasitelkiant įterptinių sistemų pagrindų sukurtų plūdurų tinklą (Gricius 2016) jį intelektualizuoti, kad jis galėtų atlikti ne tik pasyvaus stebėjimo funkcijas, bet ir būti reaktyvus, pats reaguoti į aplinkos pokyčius, bei apjungus į didesnę sistemą būtų galimas situacijų aplinkos atpažinimas bei prognozavimas.

1.2 Diskrečiųjų įvykių valdymo sistemų (DEVS) formalizavimas

1.2.1. Petri tinklų formalus aprašymas

C. A. Petri savo disertacijoje 1962 metais pasiūlė modeliavimo tinklus, kaip grafinį formalizmą leidžiantį aprašyti, modeliuoti ir valdyti sudėtingus, lygiagrečius, asinchroninius ir sinchroninius procesus. Šie tinklai labai išpopuliarėjo sėkmingai taikant juos skaičiavimo sistemų aprašymui ir įgijo Petri tinklų pavadinimą. Petri tinklai sparčiai vystomi ir dabar, kuriami standartai, jų formalizavimo ir modeliavimo priemonės taikomos daugelyje probleminių sričių, tokių kaip traukinių eismo valdymas, komunikacinių tinklinių sistemų darbo valdymas, automatizuotų valdymo sistemų, bendradarbiavimo kompiuterinių agentų planavimas ir valdymas.

Petri tinklus formaliai galima apibrėžti keturių aibių pagalba:

$N = (B, A, \Phi, H)$,.....

kur B yra baigtinė aibė simbolių, vadinamų pozicijomis $B \neq \varnothing$; A - baigtinė aibė simbolių, vadinamų perėjimais, $A \neq \varnothing$; $B \cap A = \varnothing$, $\Phi: B \times A \rightarrow \{0, 1\}$ yra tiesioginė incidentiškumo funkcija;

$H: A \times B \rightarrow \{0, 1\}$ - atvirkštinė incidentiškumo funkcija.

Nurodytos keturios aibės nusako Petri tinklo struktūrą. Paprastai Perti tinklas atvaizduojamas dviskilčiu grafu, kurio viršūnių aibė yra $B \cup A$. Pozicijų viršūnės atvaizduojamos rutuliukais, perėjimų pozicijos – vertikaliais brūkšniais. Iš pozicijos b_i eina lankas į viršūnę - perėjimą, tada ir tik tada jei $\Phi(b_i, a_j)=1$; iš perėjimo a_j į poziciją b_i eina lankas, tada ir tik tada, jei $H(b_i, a_j)=1$. Kiekvienam perėjimui $a_j \in A$ galima apibrėžti aibę įėjimo pozicijų $\Phi(a_j)$ ir aibę išėjimo pozicijų $H(a_i)$:

$\Phi(a_j) = \{b_i \in B \mid \Phi(b_i, a_j) = 1\}$

$H(a_j) = \{b_i \in B \mid H(a_j, b_i,) = 1\}$

kur $i = \{1, \ldots, n\}$; $j = \{1, \ldots, m\}$; $n = |B|$; $m = |A|$.

Analogišku būdu įvedama aibė įėjimo perėjimų pozicijai b_i - $H(b_i)$ ir aibė išėjimo perėjimų $\Phi(b_i)$ pozicijai b_i :

$H(b_j) = \{a_j \in A \mid \Phi(a_j, b_i) = 1\}$

$\Phi(b_j) = \{a_j \in B \mid \Phi(b_i, a_j,) = 1\}$

Duotas Petri tinklų apibrėžimas gali būti pritaikomas tik modeliuojamos sistemos statikos (t.y. įvykių tarpusavio ryšių ir sąlygų) atvaizdavimui. Tam, kad galima būtų atvaizduoti funkcionavimo dinamiką, įvedama tinklo pažymėjimo, žymės (markerio) funkcija:

$M: B \rightarrow \{0, 1, 2, \ldots, n\}$.

Šios funkcijos M pagalba tinklo pozicijos pažymimos sveikais teigiamais skaičiais, kurios grafiškai atvaizduojamos taškais talpinamais pozicijose ir vadinamais žymėmis.

Tinklas funkcionuoja, pereinant nuo vieno pažymėjimo prie kito pagal perėjimo apibrėžtą pozicijų sujungimo struktūrą. Pradinis tinklo žymėjimas aprašomas: $M_o = B \rightarrow \{0, 1, 2, \ldots, n\}$. Žymių pasikeitimas įvyksta vieno iš perėjimo $a_j \in A$ suveikimo rezultate.

Būtina perėjimo a_j suveikimo sąlyga yra:

$\forall b_j \in \Phi(a_j) \{M(b_i) >= 1\}$

kur $M(b_i)$ yra pozicijos b_i pažymėjimas.

Perėjimas a_j, kuriam išpildoma duota sąlyga, apibrėžiamas kaip aktyvus. Aktyvaus perėjimo a_j suveikimas pakeičia tinklo pažymėjimą $M(b) = (M(b_1), M(b_2), \ldots, M(b_n), M(b_N))$ į pažymėjimą $M'(b)$ pagal taisyklę:

$M'(b) = M(b) - \Phi(a_j) + H(a_j)$

tai yra perėjimas a_j išima po vieną žymę iš savo kiekvienos įėjimo pozicijos ir patalpina po vieną žymę į kiekvieną savo išėjimo poziciją.

Žymėjimo pasikeitimui iš M į M' tinkle pavaizduoti naudojama išraiška $M' \xrightarrow{a_j} M$

Petri tinklas, kuriame naudojama žymėjimo funkcija, vadinamas pažymėtu Petri tinklu, kuris aprašomas:

$N = (B, A, \Phi, H, M_0)$,

kur M_0 yra pradinis tinklo pažymėjimas.

Labai svarbi Petri tinklų savybė yra modelio sudarymo hierarchinių sudėtingumo lygių atvaizdavimo galimybė. Kiekvienas tinklas gali būti suprantamas

kaip makro - perėjimas arba makro - pozicija. Be to perėjimas ar pozicija gali būti detalizuotas kaip atskiras potinklis.

Petri tinklai, pagal formalų apibrėžimą pateiktą [Petri, 1960], tinka nedeterminuoto laiko intervalo įvykių aprašymui. Šiuo atveju modelis išreiškia tik įvykių pasireiškimo momentus. Petri tinklo funkcionavimas vyksta nedeterminuotai, tada kai susidaro sąlygos perėjimo aktyvizavimui (t.y. atsiranda pažymėjimas, patenkinantis vieno ar kelių perėjimų paleidimo sąlygas). Vieno perėjimo suveikimas tokiame modelyje laikomas momentiniu, nes laikomasi prielaidos, kad tikimybė vienu metu įvykti dviems ar daugiau įvykių lygi nuliui. Ši Petri tinklų savybė, neišreiškianti laiko parametrų panaudojimo, modeliuojant determinuotus procesus, yra esminių jų taikymų, modeliuojant skaičiavimo sistemas, apribojimų priežastimi.

Petri tinklų pagalba sudarytas modelis leidžia tirti labai svarbias sistemos veikimo savybes:

S1. *Pasiekiamumą*. Duotame tinkle N_M kurio pradinis žymėjimas M_0 reikia nustatyti: ar pasiekiamas pažymėjimas M' iš pradinio duoto pažymėjimo M_0. Ši modeliavimo problema interpretuojama kaip galimybė pasiekti atitinkamą sistemos veikimo būseną. Analizės metu panaudojama visų pasiekiamų tinklo N_M pažymėjimų aibės sąvoka. Ši aibė žymima $R(N_M)$.

S2. *Gyvybingumą*. Petri tinklo perėjimo gyvybingumo savybė nusako principinę galimybę, kad perėjimas suveiks, esant tam tikram pradiniam tinklo pažymėjimui M_0.. Petri tinklas patenkina gyvybingumo savybę, jeigu jo visi perėjimai yra gyvybingi. Duoto modelio gyvybingumo analizė leidžia nustatyti negyvybingus perėjimus, t.y. negalimas modeliuojamos sistemos būsenas.

S3. *Tinklo pažymėjimo apribojimus*. Pozicija b_i tinkle N_M vadinama apribota skaičiumi k, jei egzistuoja toks sveikas skaičius k, kuriam $M(b_i) \lozenge \leq k$, bet kuriam tinklo pažymėjimui $M \in R(N_M)$. Jeigu tinkle N_M, $\forall\, b_i \in B\{M(b_i) \leq k\}$, $\forall M \in R(N_M)$, tai N_M vadinamas *k* apribotu.

S4. *Tinklo apsaugojimą*. Tinklas N_M vadinamas apsaugotu, jei tenkinama sąlygą:

$$\forall M(b_i) \in R(N_M)\{\ \forall b_i \mid M(b_i) \leq 1\},\ i=\{1,\ldots, n\};\ n=|B|.$$

Kitais žodžiais tariant, apsaugotu vadinamas toks Petri tinklas, kurio bet kurioje pozicijoje, esant bet kokioms sąlygoms, negali būti daugiau nei viena žymė (t.y. tinklo apribojimas *k* lygus *1*).

Realių sistemų modelių apribojimo ir apsaugojimo savybių analizė leidžia nustatyti sistemų funkcionavimo galimybes, esant tam tikriems stacionariems režimams. Šių savybių tyrimų metu, pvz. gali būti nustatyti reikalavimai sistemos buferinėms atmintims ir pan.

S5. *Tinklo pažymėjimų išlaikymo savybė*. Kita tinklo N_M pažymėjimų apribojimo sąlyga gali būti išreiškiama:

$$\sum_{b_i \in B} M(b_i) = const,\ \text{kiekvienam}\ M(b_i) \in R(N_M),$$

arba papildomai priskiriami svoriai w_{bi} pozicijoms, ir tada ši sąlyga išreiškiama sekančiai:

$$\sum_{b_i \in B} M(b_i)\, w_{bi} = const,\ \text{kiekvienam}\ M(b_i) \in R(N_M),$$

Tinklo pažymėjimų išlaikymo savybės analizė yra svarbi, kai tiriami dinaminiai objektai (žymės) atvaizduoja nekintančius sistemos resursus.

Įprastų Petri tinklų panaudojimo patirtis parodė, kad didelio matavimo tinklo gyvybingumo, pasiekiamumo ir pan. savybių analizė reikalauja daug imlaus darbo. Ši

priežastis paskatino įvesti tam tikrus ribojimus tinklo struktūroms, kurie supaprastintų jų analizę.

Įvestas ribojimas:

$\forall a_j \in A \{\Phi(a_j) = H(a_j) = 1\}$; $j=\{1,..., m\}$; $m=|A|$

leidžia priskirti Petri tinklus grafams automatams.

Pozicijų įėjimo ir išėjimo perėjimų aibės ribojimas

$\forall b_j \in B \{\Phi(b_j) = H(b_j) = 1\}$; $j=\{1,..., n\}$; $n=|B|$ yra pažymėtų grafų poklasio požymis.

Išvystant Petri tinklų modeliavimo galimybes, buvo pasiūlyta daug jų praplėtimo variantų:

- *Apibendrinti Petri tinklai*:

$N_G = (B, A, \Phi, H, M_0)$, kuriems galioja

Φ: B X A$\rightarrow w_{\Phi} = \{0, 1, 2, ...\}$;

H: A X B$\rightarrow w_H = \{0, 1, 2, ...\}$;

Apibendrinančio tipo Petri tinklo grafas turi daugkartinio grafo formą, kuriame galimi keli lankai tarp pozicijos ir perėjimo. Šie lankai gali būti pažymėti tam tikrais svoriais. Tačiau toks Petri tinklų praplėtimas nepadidina modeliuojamųjų savybių ir principiniai apibendrinti tinklai pilnai ekvivalentūs įprastiems Petri tinklams.

Nuspalvinti Petri tinklai taip pat savo galimybėmis ekvivalentūs įprastiems tinklams, tik čia įvedama dar viena žymių nuspalvinimo aibė, kuri leidžia sumažinti sudėtingos sistemos funkcionavimą atvaizduojančio grafo struktūros sąskaita.

Prioritetiniai ir nulio patikrinimo tinklai leidžia įvertinti modeliuojamos sistemos įvykių prioritetiškumą. Tuo tikslu, nulio patikrinimo tinkluose papildomai įvedama aibė Φ tiesioginių incidentiškų uždraudimo lankų, kuriems $\Phi \cap \Phi_I = \varnothing$.

Grafe lankai atvaizduojantys aibę Φ_I vadinami uždraudimo lankais. Tokio tipo tinkle perėjimas suveiks, išpildžius dvi sąlygas:

- visose perėjimo įėjimo pozicijose $\Phi(a_j)$ turi būti po žymę;
- visose uždraudimo pozicijose $\Phi_I(a_j)$ neturi būti žymių.

Prioritetiniuose Petri tinkluose įvedama speciali prioritetiškumo funkcija, aprašanti perėjimų paleidimo prioritetus.

Sudarant modelį, dažnai tenka įvertinti modeliuojamų įvykių laiko parametrus. Tuo tikslu buvo pasiūlyti Petri tinklų praplėtimai: laiko tinklai, Merlino tinklai, E-tinklai, spalvotieji Petri tinklai (CPN).

1.2.2. Spalvotieji Petri tinklai (CPN)

Spalvotieji Petri tinklai (CPN) leidžia nagrinėti sistemą gana detaliai, nes kiekvienas sistemos žingsnis aprašomas atskirais komponentais. Kiekviename paslaugų sistemos žingsnyje nustatoma realios situacijos analizė, kurios tinkamam algoritmui sukurti taikomas atitinkamas Petri tinklų architektūros modelis, pagal kurį vykdomas paslaugos atlikimo scenarijus. Kiekviena sistemos posistemė suskaidoma į komponentus – operacijas ir būsenas. CPN modeliuoja sistemos būsenas. Ryšių kanalais perduodama informacija, reikalinga keičiant vieną būseną kita, jais atliekami valdančios informacijos perdavimai ir / arba materialiųjų srautų judėjimai. Sistemos būsena nustatoma "būsenos žymėmis", ji fiksuojama tam tikroje pozicijoje. Būsenų pokyčiai vaizduojami žymių pokyčiais. Žymių srautai atitinka objektų parametrų

pokyčius (išteklių, signalų, duomenų) ir modeliuojami sistemoje per žymių parametrų aibes bei operatorius.

Spalvotieji Petri tinklai CPN apibūdinami tokiu aibių rinkiniu (Billington et al. 2004):

CPN = (Σ, P,T , A, N,C,G, E, I);

čia Σ – baigtinė, ne tuščia spalvų (gali būti ir duomenų tipų) aibė; P – baigtinė pozicijų aibė; T – baigtinė perėjimų aibė; A – baigtinė, ne tuščia jungčių aibė; N – mazgų (transformacijų, funkcijų) aibė, siejanti kiekvieną jungtį su pora (pozicija, pereiga) arba (pereiga, pozicija); C – pozicijų spalvų funkcija, siejanti kiekvieną poziciją iš aibės P su spalva iš aibės Σ; G – pereigų kontrolės reiškinių (funkcijų) aibė; E – jungčių reiškinių (funkcijų) aibė; I – inicijavimo reiškinių (funkcijų) aibė. Aibių elementai žymimi mažosiomis aibės. M - tinklo žymėjimą realiuoju laiku nusakanti multi-aibė, apimanti pozicijose esančius žymenis, o Y - einamuosius tinklo žingsnius nusakanti aibė. Petri tinklo veikimą galima nusakyti pagal perėjimų suveikimo principus (Ding, Liu, 2008), kur pradinis tinklo žymėjimas aprašo galimybę iš pradinės būsenos pereiti į kitas būsenas: $\forall p \in P : M0(p) = I(p)$; n+1 -asis tinklo žymėjimas: $\forall p \in P : Mn+1(p) = I(p)$.

Dažnai norima atvaizduoti objektų parametrus (pvz., pavadinimą, kiekį). Tokių parametrų sudėtingesniems sąveikos būdams atvaizuoti įvedamos spalvotos žymės. CPN modelyje parametro tipą nurodo spalva. Perėjimai aprašomi taisyklėmis, kurių pagalba keičiamos gaminamų žymių spalvos. Panaudotų žymių spalvomis aprašomi ryšiai tarp įėjimo žymių ir išėjimo žymių. Taip pat galima aprašyti „prieš sąlygas“, kurios nustato naudojamų žymių spalvas ir pagal tai atpažįsta parametrų tipus bei galimas operacijas atliekamas su šiais parametrais. Spalvoti Petri tinklai suteikia galimybę žymėti skirtingomis spalvomis tiek procesus tiek ir vykstančių parametrų pokyčius ir tuo būdu atspindėti sutartines duomenų reikšmių dinamines savybes. Kiekviena pozicija gali būti susieta su tam tikrų parametrų tipų rinkiniu, kuris apsprendžia duomenų tipų rinkinio struktūrą, kurią pozicija gali saugoti ir yra toje pozicijoje išreiškiamų sąlygų įvykdymo išraiška.

CPN modelis konstruojamas taikant pozicijų (angl,. places), vaizduojamų apskritimais ir perėjimų (angl,. transitions) vaizduojamų stačiakampiais bei nukreipimo lankų grafinę notaciją. Sąveika modeliuojama kryptinėmis linijomis ir imituoja darbų, valdymo arba duomenų perdavimo srautus. Aprašas yra CPN ML programavimo kalba. Pozicijos yra naudojamos atvaizduoti sistemos būklę. Kiekviena vieta galima pažymėti vienu ar daugiau žymių, ir kiekviena žymė neša savyje duomenų vertę. Ši duomenų vertė yra vadinama žymės spalva. Tai yra žymių skaičius ir žymių spalvos ant individualių vietų, kurios tuo pačiu metu atvaizduoja sistemos būseną. Tai vadinama CPN modelio žymėjimu. Šalia kiekvienos pozicijos yra aprašas, pagal kurį nustatomas spalvų rinkinio požymis (duomenų vertės). Spalvų vertės yra apibrėžtos naudojant CPN ML raktažodžiu - „colset“ ir spalvos vertė - NO yra apibrėžta visoms sveikųjų skaičių INT reikšmėms.

CPN leidžiama atlikti interaktyviąją imitaciją, kurios metu rezultatai pateikiami tiesiogiai diagramoje. Imitacijoje galime lengvai sumodeliuoti didelę ir sudėtingą sistemą, kurioje galima stebėti ir reikalui esant keisti judančių žymių „pernešamą“ informaciją. Taipogi leidžiama atlikti sistemos būsenos analizę, t.y. nustatyti konfliktuojančias būsenas, konkurencinius veiksmus, sinchronizacijos problemas, pasikartojančių būsenų ir bendrų resursų panaudojimo galimybes. Fiksuoti gaunamų parametrų sekas belaidžio tinklo protokole, įvedami papildomi parametrai, numatomi SPN aprašyme: colset - spalvų rinkinys, NO = int - spalvą atitinkančio

parametro tipo numeris. NO naudojamas modeliuoti veiksmų sekos eiliškumą protokole.

Spalvų rinkinio ir duomenų Dekarto sandauga NOxDATA išreiškia gaunamų duomenų ir jų prieskyrų skirtingiems duomenų tipams sąveiką. Čia pirmas narys yra pozicijos eilės numeris, antrasis narys fiksuojami duomenys išreiškiami teksto eilute.

Pagal (Homepage of the CPN Tools, 2011) spalvų rinkiniai aprašomi pagrindiniais parametrais taip:

colset DATA = string;

colset NOxDATA = product NO * DATA.

Spalvos rinkinys DATA yra naudojamas, kad modeliuoti naudingąją duomenų paketų apkrovą ir yra apibrėžiami visų teksto eilučių (String) rinkiniu. Spalvų rinkinys NOxDATA naudojamas modeliuoti duomenų paketus, kur taikomi duomenys yra eilės numeris ir patys duomenys.

1.3 Neuroninių tinklų taikymo poreikis

Mūsų nagrinėjamose sistemose galima būtų panaudoti neuroninius tinklus norint nustatyti esame aplinkos būsena, bei bandyti nustatyti ateities būseną. Norint sukurti ir apmokyti neuroninį tinklą nagrinėjamai sistemai, pirmiausia reikia sukaupti duomenis apie esamą sistemą , sukaupti aplinkos būsenų analizę kurią turėtų atlikti tos srities specialistai ir išskirti būsenų kitimus, apibrėžti taisykles kada yra normali aplinkos būsena, ir kada ji pakinta iš įprastos į speficinę kuria reikia identifikuoti. Šiam identifikavimui, ir modelaivimui reikėtų sukurti ir apmokyti tris skirtingus neuronų tinklus.

Pirmasis apmokytas neuroninis tinklas yra kuriamas, kad gavus iš pradinių reikšmių pačiame plūdure pagal koeficientus, būtų galima apskaičiuoti pačiame plūdure sistemos pakitimą pagal jame esančius duomenis.

Antras neuroninis tinklas būtų skirtas duomenimis kurie būtų surinkti iš visų plūdurų ir papildomų sistemų kurie būtų analizuojami ir galėtų turėdami daugiau įvedamų parametrų nustatyti būsenos pakitimą kurį galima aptikti tik lyginant daugelių agentų(plūdurų) tiekiamus duomenis.

Trečiasis neuroninis tinklas būtų kuriamas , kuris skirtas numatyti galimas ateities reikšmes, pavyzdžiui padidėjęs drumzlėtumas tam tikruose agentuose gali nulemti , kad pagal vėjo kryptį ar vandens srovės, drumzlėtumas bus nuneštas į kitą geografinę padėti, kas gali padėti apskaičiuoti, kada tai pasieks krantą , ir kada gali reikėti pradėti valymo darbus pakrantėje.

Ši tinklų logika , kad neuronų tinklas gali atlikti šias tris funkcijas yra aprašyta (Garcia-Alba et all, 2019) ir detaliai parodyta, kad pakankamai patikimai galima numatyti E.Coli užterštumą.

1.4 Daugiaagentinių sistemų sąveikos modelių poreikis

Pluduro sistema sudaryta iš keletos sensorinių, jutiklinių, mikrovaldiklinių įterptinių sistemų. Pludure gali būti net 24 skirtingi tipai tokių posistemių. Programiniame lygmenyje skirtingų sistemų darbui užtikrinti tinkamiausias būdas būtų pasitelkti daugelio agentų sąveikos metodus (Gricius et al., 2014)

Kadangi kiekviename plūdure dirba keletas skirtingų sensorių su atskiro tipo paskirtimis, jo veikimas gali būti aprašomas kaip keletos programinių agentų veikimas. Atskiras agentas bandydamas nustatyti atitinkamo parametro reikšmę ir įvertinti tam tikrą aplinkos būseną, vykdo parametrų fiksavimą ir perduodą tuos duomenis visai sistemai. Turime nagrinėti sistemą kuri yra sudaryta iš daugelio agentų. Tokios sistemos, kuriose veikia keletas skirtingas funkcijas atliekančių autonominių yra vadinamos daugiaagentinėmis sistemomis (DAS).

Kadangi agentas gali gauti tik savo paskirties duomenis ir juos paversti aplinkos įvertinimo informacija, jo sąveika su artimų agentų informacija taip pat svarbi. Kiekvienas agentas turi tik ribotos paskirties informaciją, mums reikia visus agentus sujungti į bendrą sistemą.

Taip pat yra agentai kurie interpretuoja kitų šaltinių matuojamus duomenis ir paverčia ją informacija, kuri perduodama bendrai sistemai. Daugiaagentinės sistemos aplinkos stebėjimo procesuose, kurios stebi vandenį ir meteorologinius duomenis susijusius su vandens telkiniais dažniausiai susitelkia ties vienu kažkokiu parametru ir analizuoja būtent tą parametrą nekreipdami dėmesio į kitus. Kai kuriuose darbuose nagrinėjama, geriamojo vandens mikrobinė taršą (Hansen et all., 2019), kituose eutrofikacija ežeruose (Janssen, Carpenter, 2019) arba atlieka vertinimus, matuodami tik vieną specifinę mažos apimties vandens analizės sistemą (Sharma 2019).

1.5 1 skyriaus išvados

Apžvelgti tik keli daugiagentinių sistemų projektavimo metodai. Tačiau tinkamų dirbtinio intelekto metodų paieškos dar turėtų būti tęsiamos.

2 REIKALAVIMAI VANDENS TARŠOS STEBĖSENOS SISTEMAI UŽTIKRINTI

Reikalavimus stebėsenos sistemos kūrimui būtų galima išskirti į kelias skirtingas dalis, tai yra:

- bendros visos sistemos veikimo reikalavimai, kur yra įtraukiami atskiros pludurų sistemos, belaidžiai tinklai, serveriai, duomenų saugyklos, bei visi pagrindiniai skaičiavimų resursai;
- antroji dalis būtų, atskiro plūduro kurio veikimas grįndžiamas įterptinių possistemių architektūra, nes nuo to kaip veiks plūduras, priklausys kokias galimybes jis turės, ir kaip veiks visa sistema.

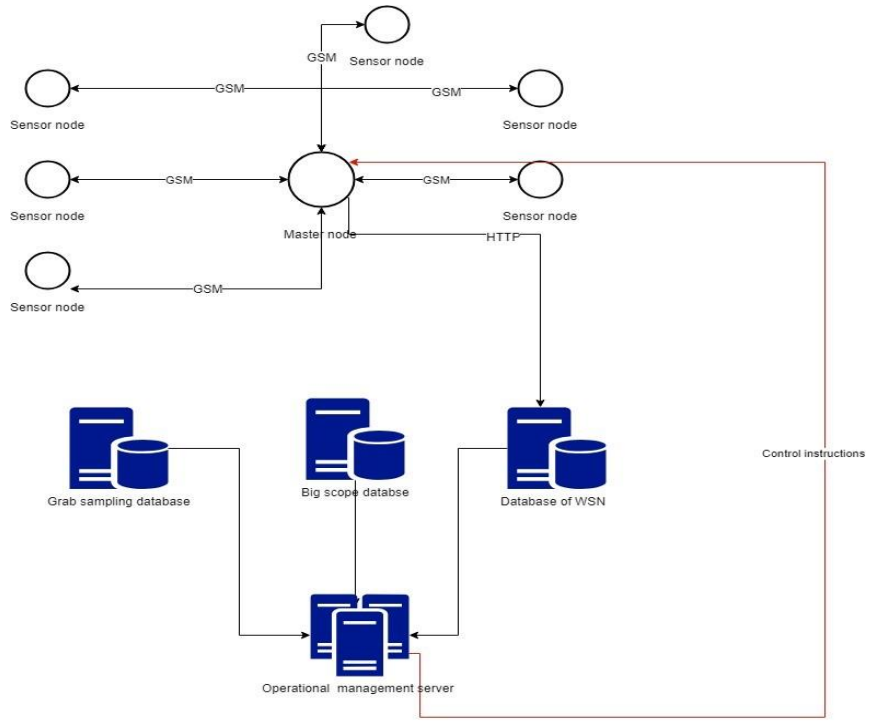
Sistema turi prisitaikyti prie bendrų belaidžio tinklos ir išskirstytū komponentų galimybių, kurios gali būti kintančios.

2.1. Reikalavimai visos sistemos architektūrai belaidžių tinklų infrastruktūroje

Sistema turėtų būti sudaryta iš kelių atskirų posistemių, kurios veikia kartu ir atskirai viena nuo kitos, bei kartu netiesiogiai kai surinkti duomenys yra analizuojami bendrame kontekste. Kadangi tiriama sistema turi veikti vandenyje, atsiranda pirmiausias skirstymas sistemų pagal vandens telkinio tipą, nes tai nurodo kokiu ryšiu gali būti pasiekiamas mūsų sensorių plūduras:

- Atviri dideli vandens telkiniai (jūra)
- Vidaus vandenys (upės, ežerai)

Bendros architektūros elementai jungiami į sistemą turėtų susidaryti iš 6 dalių.



Paveikslas 1. Jūros vandens stebėsenai siūloma sistemos architektūra

Jūros užterštumas yra dažniausiai veikiamas didesnių taršos kiekių, kurie gali būti atnešami iš kitų regionų , bei aktyvios laivybos Lietuvos pakrantėje daromos įtakos. Mūsų nagrinėjama sistema labiausiai galėtų būti pritaikyta, jūros bei upių stebėjimui. Tokių vandens telkinių svarba išauga vien dėl didesnio žmogaus poveikio joms. Upės yra veikiamos stipriai agrokultūros, nes paviršiniai vandenys dažniausiai suteka per mažus upelius į didesnes upes, ir taip pasiekia kuršių marias, bei dauguma vandens valymo įrenginių išteka į upes, kur atsiranda poreikis stebėti ar nėra padidėjusių tam tikrų parametrų kurie galėtų indikuoti padidėjusį užterštumą.

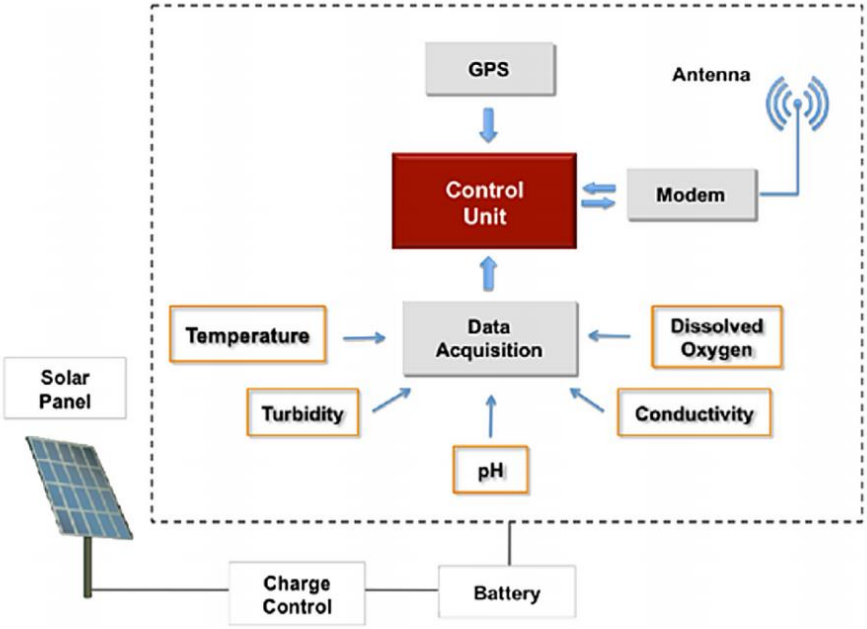
Stebėsenai organizuoti siūloma sistemos architektūra, pateikta 1 paveiksle, kurios pagrindinai komponentai yra:

1. Yra nuotolinių sensorių (plūdurų) tinklas kuris matuoja duomenis ir siunčia pagrindiniam mazgui.
2. Pagrindinis mazgas, yra atsakingas už taisyklingą duomenų surinkimą iš sensorių tinkle, mazge patikrinama informacija ar jos laiko žymė atitinka tą kurios buvo tikėtasi, ieškoma neatitikimų duomenyse, atliekama duomenų validacija ir persiunčiama toliau į duomenų bazę kurioje yra saugomi visi istoriniai sensorių tinklo duomenys
3. Palydovinių nuotraukų apdorojimo sistema, kuri analizuoja palydovines nuotraukas, bei išsaugo reikiamus duomenis į duomenų bazę tolimesnei kombinuotai analizei.
4. Mėginio ėmimo sistema, tai sistema kuri renka iš viešai prieinamų šaltinių (sveikatos centro duomenų bazių, sveikatos apsaugos portalo bei kitų internetinių platformų) ir mėginių ėmimo atliktų mokslininkų kurie turi didesnį tikslumą ir patikimumą.
5. Didelių duomenų analizės sistema, kuri yra skirta susisteminti didelius kiekius duomenų , bei joje yra aprašytos taisyklės kuriomis yra apjungiami skirtingų tipų bei laiko duomenys.
6. Žiniomis grindžiama sistema, kuri analizuoja duomenis, ir naudodamasis intelektualizavimo metodais(neuroniniai tinklai, mašininis mokymasis , žinių bazėmis) identifikuoja esama situacija, bei bando prognozuoti galimas būsimas sistemos reikšmes.

2.2. Vandens parametrų stebėsenai taikomi sensorių tipai ir jų sandara

Kiekvienas plūduras elgtųsi kaip atskiras agentas turintis pagrindinį tikslą, išmatuoti duomenis ir juos persiųsti bei gauti atsakymą, kad jo duomenys buvo gauti dėl to yra svarbu, išskirti kaip turėtų būti sudarytas plūduras, kad mūsų sistema galėtų patikimai jais naudotis.

Sensorių tinklas susidaro iš mažų plūdurų kurie gali dirbti autonomiškai apsirūpinti energija iš baterijų ar saulės panelių kurios generuoja energiją. Kiekviename plūdure yra ryšio komunikacijai skirta dalis kuri gali būti tiek ZigBee modulis, tiek GSM kuris gali būti parenkamas, pagal plūdurų išsidėstymą geografinėje padėtyje vienas su kitu, bei su bendra geografine padėtimi, nes plūdurai esantys jūroje turi turėti būtinai GSM ryšį, kai upėje esantiems plūdurams gali užtekti ZigBee ryšio aprėpties. GSM ryšio modulį turintį plūduras yra detalizuojamas paveiksle 2.



Paveikslas 2. Pavysdys struktūrinių komponentų reikalingų plūduro konstravimui, Šaltinis: sudaryta pagal (Newton, et al., 2016)

2.3. Mikro-valdiklių ir jutiklių tinklo projektavimas ir kūrimas

Pateiksime pavyzdį, kaip galima sukurti ryšį ir vykdyti komunikaciją tarp tinklo valdiklių ir jutiklių, projektuojant ir realizuojant tai *Arduino-Atmega* plokštės pagrindu ir gauti reikiamus duomenis prijungiant tam tikrus jutiklius.

2.3.1. Komunikacijos tarp mikrokontrolerių technologinės platformos pavyzdys

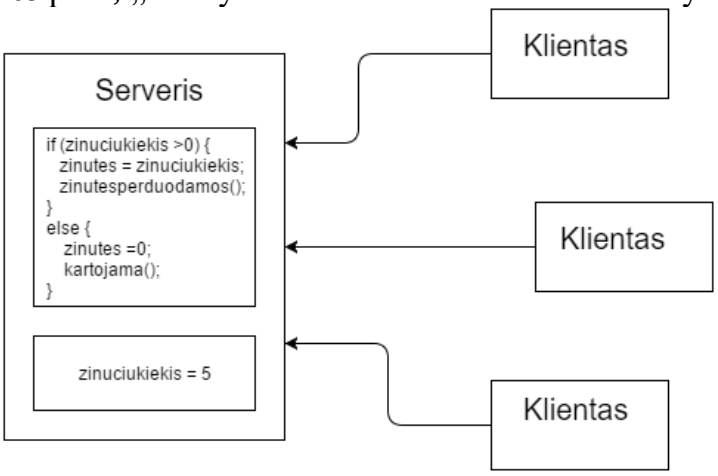
Arduino – tai prijungiama prie kompiuterio platforma, naudojanti *Atmel* firmos mikrokontrolerius, turinčius itin didelių galimybių. Ši atviro kodo platforma tapo populiari visame pasaulyje. Dar vienas *Arduino* privalumas – jam yra sukurti papildomi skydeliai (angl. *Shield)*, kurie suteikia galimybę *Arduino* prijungti prie interneto, *WiFi* tinklo, *Bluetooth* ryšio, taip pat naudoti SD atminties korteles ir dar daug visokių kitų galimybių. *Arduino* platformoje gali būti įdiegti įvairūs įrenginiai − nuo telefonijos įrangos iki smulkiausių jutiklių, matuojančių aplinkos parametrus.

Pasinaudojus *javaComm* programavimo sąsaja galima sukurti programinę įrangą jutiklių duomenis konvertuoti į tekstines žinutes.

2.3.2 Mobiliojo ryšio tarp kliento ir serverio struktūra ir komponentų programavimas

2.3.2.1. Kliento ir serverio ryšio schema

Kliento ir serverio ryšio schemoje (2.1.1 pav.) pavaizduota kliento pusė ir serverio pusė. Klientas aktyvus ir užmezga ryšį su serveriu. Serveris yra dažniausiai yra pasyvus, t. y. laukia, kol klientas užmegs ryšį su juo. Komunikacijai užtikrinti klientas ir serveris sukuria programinę jungtį per jungtis (angl. *socket)*. Dažniausiai naudojamos TCP/IP ir UPD/IP jungtys. Kliento ir serverio komunikacijai užtikrinti vartosime objektiškai orientuotas programavimo kalbas *Java* ir C#. Pagrindinės klasės, įgyvendinančios tinklinį programavimą, realizuotos pakete *java.net*. *Java* ryšys tarp dviejų įrenginių tinkle panašus į įprastą darbą su įvedimo ir išvedimo srautais, tik šiuo atveju srautai nukreipiami tarp įrenginių jungtimis. Serverio pusė laukia, kol prie jos pasijungs kliento pusė, „klausydama“ tinklo tam tikrame nustatytame prievade.



Paveikslas 3. Kliento ir serverio ryšio schema

TCP yra vienas iš pagrindinių protokolų, esančių Internetinių protokolų rinkinyje (angl. *Internet protocol suite)*, sudaro tarpinį lygį tarp IP ir taikomosios programos bei priklauso transportavimo lygmeniui. Naudodami šį protokolą serveris ir klientas sukuria jungtis, procedūras tarp kelių tinklo taškų (įrenginių) ir dalijasi duomenimis. Priešingai nei UDP, šis protokolas užtikrina patikimą duomenų perdavimą tarp dviejų tinklo taškų.

Serverio pusės procesams nustatyti reikia penkių žingsnių.

1. Sukurti *ServerSocket* objektą.

Inicijuojant *ServerSocket* objekto konstruktorių, reikalaujama serverio prievado (angl. *port)*. Pirmiausia sukuriama programinė jungtis panaudojant *ServerSocket* objektą, kuriame yra šios jungties aprašas. *ServerSocket* objektas atlieka tinklo klausimo funkcijas, kurios rezultatas − jungtis tarp skirtingų įrenginių. Serverio prievadas yra sveikas skaičius, nusakantis prievado numerį, kuris gali būti 1024−65535, išskyrus rezervuotus skaičius. *ServerSocket* objekto sukūrimas:

ServerSocket serverSocket = new ServerSocket(1234);

šiuo atveju serveris laukia („klausosi“) komunikacijos iš kliento pusės prievado skaičiaus 1234.

2. Serverio laukimo būsena.

Serveris laukia neribotą laiką („blokuoja“kliento sujungimą). Užmegzto ryšio aprašo vertę serveris gauna taikydamas metodą *accept()*. Kviečiant metodą *accept* klasėje *ServerSocket* gaunamas *Socket* objektas ir komunikacija yra atlikta. Kol negaunamas *accept*, *Socket* objekto (programos) veikimas blokuojamas ir laukiama, kol prisijungs klientas. *Socket* objekto sukūrimo pavyzdys:

Socket linkas = serverSocket.accept().

3. Įėjimo ir išėjimo duomenų srautai.

Duomenims įvesti ir išvesti naudojami srautai (angl. *streams)*. *Java* apibrėžia tris standartinius srautus: standartinio įvedimo *System.in*, standartinio išvedimo *System.out* ir klaidų išvedimo System.err. Serveris išsiunčia ir priima duomenų srautus pasinaudodamas metodais *getInputStream* ir *getOutputStream* iš klasės *Socket. Java* įvedimo ir išvedimo klasės yra sujungtos pakete *java.io*. Gali būti baitų ir simbolių *(Unicode* simbolių) srautai.

Duomenų nuskaitymui naudojamas *Scanner* objektas, o kviečiant metodą − konstruktorius su parametro tipu *InputStream*. Srautams atidaryti paprastai tinka konstruktorius *Scanner (InputStream* srautas). Duomenų mainams su rinkmenomis, didelių failų, tinklo nuskaitymui dažniausiai naudojamos baitų rinkmenos. Šiuo konstruktoriumi *Scanner* objektas gauna duomenis iš sukurto *InputStream* objekto duomenų tipo srauto. *InputStream* objektas gaunamas kviečiant metodą *getInputStream.*

Scanner srautasIvestis = new Scanner(linkas.getInputStream()).

Išvesti tekstui rekomenduojama naudoti simbolių srautus. Iš *Socket* objekto gaunamas baitų įvedimo srautas, kuris pertvarkomas į simbolizuotą srautą. Gaunamus baitų srautus galima pertvarkyti į simbolių srautą taikant objektą *OutputStream*. Simbolinių srautams rekomenduojama naudoti klasę *PrintWriter,* kurios objektai sukuriami keliais alternatyviais konstruktoriais. *PrintWriter (OutputStream* srautas, *boolean b)* konstruktorius panaudojamas norint gauti simbolių srautą iš baitų srauto. Jei *b* reikšmė yra *true*, srautas išvedamas automatiškai, kai suranda pabaigos eilutę. Gali nutikti, kad duomenys nebus išvesti, jei buferis nebus pripildytas, jeigu programose išvedant nedidelius duomenų kiekius ar nuskaitant iš tinklo duomenis dalis duomenų buvo prarasta.

PrintWriter SrautasIsvestis = new PrintWriter(linkas.getOutputStream(),true).

4. Duomenų siuntimas ir priėmimas.

Pasinaudojant *Scanner* ir *PrintWriter* objektais lengviau siųsti ir priimti duomenis. Duomenų siuntimui taikomas *nextLine()* metodas, o išsiuntimui − *println()* metodas.

SrautasIsvestis.println("Laukiama duomenų…");

String srautasD = srautasIvestis.nextLine();

5. Komunikacijos uždarymas.

Baigęs apsikeitimą duomenimis, serveris nutraukia ryšį metodu *close()*. Paprastai ryšys tarp kliento ir serverio užmezgamas taikant TCP arba UDP protokolus ir 32 bitų (IPv4) arba 128 bitų (IPv6) adresus. Ryšio nutraukimas:

linkas.close();

Svarbu atkreipti dėmesį, kad dirbant su šio paketo klasėmis dažnai tenka naudoti *try{} ... catch* konstrukciją ir išimtis *throws*.

try {

vykdomas kodas;

```
} catch (tipas vardas) {
    Vykdomas kodas įvykus klaidai
}
```

Darbas su duomenų skaitymu ir rašymu yra viena iš jautresnių programos funkcijų ir tokiais atvejais numatomas išimčių apdorojimo mechanizmas.

Dirbant tinkle gali netikėtai nutrūkti ryšys, bet programa turi išlikti gyvybinga. Tokiomis kritinėmis situacijomis, kai negalima toliau tęsti normalios programos eigos, valdymas perduodamas išimtims. Tipinis pavyzdys: norime nuskaityti failo duomenis, bet nerandame jo prieigos arba panaudojama tuščio *(null)* failo prieiga. Jei kuriant programą nebuvo numatytas tokios situacijos sprendimas, programos vykdymas bus sustabdytas. Jei programoje buvo numatytas specialus išeities atvejo kodas, kuris leidžia toliau tęsti programos vykdymą, tai programa išliks gyvybinga ir tęs darbą.

Net jei problemos sprendimas nenumatytas, galima parašyti išimties apdorojimo kodą, skirtą išsaugoti tuo momentu sukauptą naudingą informaciją ir atlaisvinti užsakytus kompiuterio išteklius. *ServerSocket* nebūtina dėti į *try* bloką. Nepavykus sukurti *ServerSocket* objekto konstruktoriumi, pagrindinis metodas *(main)* generuotų išimtį ir programos darbą sustabdytų. Metodas *main* generuotų *IOException* išimtį, kuri inicijuotų programos pabaigą. Kuriant objektą *Socket* reikia jį įtraukti į *try-finally* bloką, nes baigiantis programai reikia nutraukti (užverti) jungtį tarp kliento ir serverio. Priešingu atveju, jei nenaudosime *try-finally*, bus nutraukta programa, bet ne jungtis, t. y. nebus užveriamas *ServerSocket* objektas.

Vykdoma serverio pusės programa, kurios užduotis − nuskaitinėti kliento siunčiamą duomenų srautą:

```
import java.net.*;
import java.io.*;
public class Serveris
{
 public static void main(String[] args) throws IOException
  {
   ServerSocket serverSocket = null;
   try {
       serverSocket = new ServerSocket(1234); (1)
      }
   catch (IOException e)
      {
       System.err.println("klaida negalima nuskaityti prievado: 10010.");
       System.exit(1);
      }
   Socket clientSocket = null;
   System.out.println ("laukiama komunikacijos.....");
   try {
```

```
        clientSocket = serverSocket.accept();
       }
     catch (IOException e)
       {
        System.err.println("patvirtinimo klaida.");
        System.exit(1);
       }
     System.out.println ("Sujungimas sskmingas");
     System.out.println ("Laukiama duomenu.....");
     PrintWriter out = new PrintWriter(clientSocket.getOutputStream(),
                                true);
     BufferedReader in = new BufferedReader(
           new InputStreamReader( clientSocket.getInputStream()));
     String inputLine;
     while ((inputLine = in.readLine()) != null)
       {
        System.out.println ("Serveris: " + inputLine);
        out.println(inputLine);
        if (inputLine.equals("pabaiga"))
           break;
       }
     out.close();
     in.close();
     clientSocket.close();
     serverSocket.close();
    }
}
```

Kliento pusės jungčiai palaikyti sukuriamas klasės *Socket* objektas. Šio objekto konstruktoriui reikia nurodyti norimo serverio IP adresą ir prievado numerį. Kada jungtis sėkmingai sudaryta, skirtumo tarp serverio ir kliento pusės nebelieka ir atliekami duomenų mainai. Kaip ir serverio pusėje, kliento pusėje duomenų mainams naudojami tie patys metodai *getInputStream* ir *getOutputStream* sukurti baitų srautus *InputStream ir OutputStream*. Kaip minėta, juos galima įtraukti į simbolinius srautus.

Kliento pusės turi žinoti galinio įrenginio IP adresą arba serverio vardą, arba jo DNS adresą. Galimybė gauti statinę klasę *InetAddress* metodu *getByName (String* vardas) arba *getByName (String* DNS adresas). Jei numatome tik testuoti serverio ir kliento pusę tame pačiame serveryje, vartojame *localhost* kaip vardą *InetAddress.getByName("localhost"),* kuris atitinka 127.0.0.1 IP adresą arba *null* ieškant pagal vardą *InetAddress.getByName(null). Socket* (kliento pusėje) objektams kurti reikia žinoti jungties prievado numerį, koks yra nurodytas *ServerSocket* (serverio pusėje). Prievadą serverio pusėje galima įsivaizduoti kaip loginę jungtį. Visi duomenys į serverį patenka per tą pačią jungtį ir toliau skirstomi į

prievadus pagal jų numerius, ateinančius kartu su duomenų srautu. Prievadai naudojami kompiuteriui palaikant daugelį serverių, iš to išplaukia reikalavimas pateikti ne tik IP adresą, bet ir prievado numerį.

```
public class Klientas {
   public static void main(String[] args) throws IOException {

      String serverHostname = new String ("127.0.0.1");

      Socket echoSocket = null;
      PrintWriter out = null;
      BufferedReader in = null;

      try {
      echoSocket = new Socket(serverHostname, 1234);
         out = new PrintWriter(echoSocket.getOutputStream(), true);
         in = new BufferedReader(new InputStreamReader(
                              echoSocket.getInputStream()));
      } catch (UnknownHostException e) {
         System.err.println("hostas nerastas: " + serverHostname);
         System.exit(1);
      } catch (IOException e) {
         System.err.println("negaunamas IO" + serverHostname);
         System.exit(1);
      }

   BufferedReader stdIn = new BufferedReader(
                           new InputStreamReader(System.in));
   String userInput;

      System.out.print ("irasykite: ");
   while ((userInput = stdIn.readLine()) != null) {
      out.println(userInput);
      System.out.println(": " + in.readLine());
         System.out.print ("irasykite: ");
   }
   out.close();
   in.close();
   stdIn.close();
```

```
    echoSocket.close();
    }
}
```

Kaip minėta, TCP yra tarpinis lygis tarp IP ir aplikacijos, priklausantis transportavimo lygmeniui. Naudodamos šį protokolą, taikomosios programos gali sukurti procedūras tarp kelių tinklo taškų ir dalintis duomenimis. Priešingai nei UDP, šis protokolas užtikrina patikimą duomenų perdavimą tarp dviejų tinklo taškų.

2.3.3. UDP serverio darbo schemos pavyzdys

UDP (angl. *User Datagram Protocol)* yra TCP/IP naudojamas perdavimo protokolas, alternatyva TCP protokolui. Skirtingai nei TCP, UDP nėra patikimas, neatlieka duomenų tėkmės kontrolės (angl. *flow control)* ir neturi klaidų atitaisymo mechanizmų, todėl naudojamas tik persiųsti duomenis. TCP atveju klientas pirmiausia užmezga ryšį su serveriu ir tik tada siunčia duomenis. Dėl to duomenų persiuntimas yra patikimas.

Gavėjas visada gaus jam adresuojamus duomenis. Bet nesant gero ryšio, kai duomenys keičiami lėtai, TCP protokolas nėra naudingas. Jei duomenų (pavyzdžiui, vaizdo ar telefonijos) perdavimas vyksta realiu laiku, teikiamo adresato nepasiekusi žinutė bus ignoruojama, nes neturės prasmės jos persiuntimas vėliau. Todėl perduodant realius duomenis dažniausiai pasirenkamas „lengvesnis“ UDP protokolas, duomenis siunčiantis dalimis. UDP protokolu siunčiama žinutė į serverį, net nežinant, ar šis serveris sugebės ją priimti. Serveris, gavęs žinutę iš kliento, žinos iš kur buvo siunčiama žinutė. Serveris sukuria programinę jungtį pasinaudodamas *DatagramSocket* objektu. Kreipiantis į objekto konstruktorių reikalaujama prievado numerio. Kaip ir TCP protokolu užmezgus ryšį, galioja tos pačios taisyklės ir pasirinkimo sąrašas tas pats − 1024−65535, išskyrus rezervuotus skaičius. Pirmiausia sukuriama programinė jungtis pasinaudojus *DatagramSocket* objektu, kuriame yra šios jungties aprašas. Programos pirminis tekstas toks:

DatagramSocket serverisSocket = new DatagramSocket(1234);

DatagramSocket nustato aktyvų ryšį su programine jungtimi ir yra pasirengęs priimti žinutes iš kliento. Jungties stebėjimo ir ryšio užmezgimo žingsniai, kurie yra būtini naudojant TCP protokolą, Praleidžiami.

Kliento pusės jungčiai sukurti naudojamas *DatagramSocket* objektas, bet neįrašant prievado numerio vyksta kreipimasis į tuščią konstruktorių, kuris atrodo taip:

DatagramSocket klientasSocket = new DatagramSocket();

Duomenų priėmimui pirmiausia reikia sukurti objektą *DatagramPacket. Datagram* paketai naudojami siekiant įgyvendinti paketų pristatymo paslaugą. Šiai paslaugai inicijuoti reikalaujama kreiptis į konstruktorių, kuriame turime pateikti gaunamų duomenų bitų masyvą ir gaunamų duomenų ilgį:

DatagramPacket g_packetai = new DatagramPacket(g_duomenys, g_duomenys.length);

Datagram objekto paketai yra naudojami siekiant įgyvendinti paketų pristatymo paslaugą jungtimis. Kiekvienas pranešimas, nukreiptas iš kliento į serverį, grindžiamas tik informacija, kuri yra tik tame viename pakete. Jei serveris priima žinutę, jis turi galimybę

nustatyti siuntėją – klientą. Keli siunčiami paketai iš vieno kliento į serverį gali būti pristatomi bet kuria tvarka. Be to, paketų pristatymas nėra garantuotas. Numatytas serveris nebūtinai priima kliento išsiųstą žinutę. Klientas, siunčiantis žinute UDP protokolu, neužmezga ryšio su serveriu taip, kaip siųsdamas TCP protokolu, bet siunčia žinutę tiesiogiai.

Duomenų išsiuntimui reikia sukurti objektą *DatagramPacket*. Paketų išsiuntimo paslaugai inicijuoti reikalaujama kreiptis į konstruktorių, kuriame turime pateikti išsiunčiamų duomenų bitų masyvą, jo ilgį, IP adresą ir prievado numerį:

```
DatagramPacket siusti_paketa =
  new DatagramPacket(siusti_duomenis, siusti_duomenis.length, IPAdresas, 1234);
```

Duomenys į serverį siunčiami pasitelkus metodą *send*():

```
klientasSocket.send(siusti_paketa);
```

UDP protokolo išeities kodas:

```
import java.io.*;
import java.net.*;
class UDPServeris {
 public static void main(String args[]) throws Exception
  {
   try
   {
    DatagramSocket serverisSocket = new DatagramSocket(1234);
    byte[] receiveData = new byte[1024];
    byte[]siusti_duomenis = new byte[1024];
     while(true)
     {
      receiveData = new byte[1024];
      DatagramPacket receivePacket =
        new DatagramPacket(receiveData, receiveData.length);
      System.out.println ("Waiting for datagram packet");
      serverSocket.receive(receivePacket);
      String sentence = new String(receivePacket.getData());
      InetAddress IPAddress = receivePacket.getAddress();
      int port = receivePacket.getPort();
      System.out.println ("From: " + IPAddress + ":" + port);
      System.out.println ("Message: " + sentence);
      String capitalizedSentence = sentence.toUpperCase();
      sendData = capitalizedSentence.getBytes();
      DatagramPacket sendPacket =
        new DatagramPacket(sendData, sendData.length, IPAddress,
```

```
                    port);
            serverSocket.send(sendPacket);
          }
        }
         catch (SocketException ex) {
          System.out.println("UDP Port 9876 is occupied.");
          System.exit(1);
         }
        }
      }
```

Duomenų srauto sekimui ir paketų analizei naudojamas „Wireshark“ paketų ir protokolų analizatorius. Naudojant šią programą galime įrašyti ir žiūrėti paketus keliaujančius tinkle, matyti jų turinį ir nustatyti protokolo tipą. Įrankis, leidžiantis stebėti žinučių siuntimo būklę realiame laike ir atvaizduojantis jų siuntimo seką.

2.4. 2 Skyriaus išvados

Skyriuje pateikta architektūra stebėsenos sistemos ir pagrindinių sistemos nutolusių klientų bei serverių sąsaja, kurios programinis kodas parašytas Java kalbos pagrindu. Norint apjungti skirtingų tipų įrenginius ir užtikrinti komunikaciją tarp jų, reikalaujama skirtingų transportavimo protokolų. Skyriuje pateikiamas programinis kodas, kuris realizuoja pagrindinius transportavimo protokolus, tokius kaip TCP, UDP ir TLS. Skyriuje aprašytas programinis kodas leidžia integruoti įrenginius į bendrą sistemą duomenų apsikeitimui.

Bendrosios išvados

Literatūra

Transforming Our World: The 2030 Agenda for Sustainable Development. (2018). In *A New Era in Global Health*. https://doi.org/10.1891/9780826190123.ap02

Baqar, M., Sadef, Y., Ahmad, S. R., Mahmood, A., Li, J., & Zhang, G. (2018). Organochlorine pesticides across the tributaries of River Ravi, Pakistan: Human health risk assessment through dermal exposure, ecological risks, source fingerprints and spatio-temporal distribution. *Science of the Total Environment*. https://doi.org/10.1016/j.scitotenv.2017.10.234

Belikova, V., Panchuk, V., Legin, E., Melenteva, A., Kirsanov, D., & Legin, A. (2019). Continuous monitoring of water quality at aeration plant with potentiometric sensor array. *Sensors and Actuators, B: Chemical*, 854–860. https://doi.org/10.1016/j.snb.2018.11.153

Dzemydiene, D., Maskeliunas, S., Dzemydaite, G., & Miliauskas, A. (2016). Semi-Automatic Service Provision Based on Interaction of Data Warehouses for Evaluation of Water Resources. *Informatica (Netherlands)*. https://doi.org/10.15388/Informatica.2016.107

García-Alba, J., Bárcena, J. F., Ugarteburu, C., & García, A. (2019). Artificial neural networks as emulators of process-based models to analyse bathing water quality in estuaries. *Water Research*, 283–295. https://doi.org/10.1016/j.watres.2018.11.063

Gricius, G., Drungilas, D., Andziulis, A., Dzemydiene, D., & Voznak, M. (2014). SOM based multi-agent hydro meteorological data collection system. *Advances in Intelligent Systems and Computing*. https://doi.org/10.1007/978-3-319-07401-6_4

Gricius, G., Drungilas, D., Andziulis, A., Dzemydiene, D., Voznak, M., Kurmis, M., & Jakovlev, S. (2015). Advanced approach of multiagent based buoy communication. *Scientific World Journal*. https://doi.org/10.1155/2015/569841

Han, Z. G., & Cui, C. H. (2009). The application of zigbee based wireless sensor network and GIS in the air pollution monitoring. *Proceedings - 2009 International Conference on Environmental Science and Information Application Technology, ESIAT 2009*. https://doi.org/10.1109/ESIAT.2009.192

Hansen, C. B., Kerrouche, A., Tatari, K., Rasmussen, A., Ryan, T., Summersgill, P., … Albrechtsen, H. J. (2019). Monitoring of drinking water quality using automated ATP quantification. *Journal of Microbiological Methods*, *165*(May), 105713. https://doi.org/10.1016/j.mimet.2019.105713

Hart, J. K., & Martinez, K. (2006). Environmental Sensor Networks: A revolution in the earth system science? *Earth-Science Reviews*. https://doi.org/10.1016/j.earscirev.2006.05.001

Hoagland, P., & Scatasta, S. (2006). The Economic Effects of Harmful Algal Blooms. In *Ecology of Harmful Algae*. https://doi.org/10.1007/978-3-540-32210-8_30

Keshtgari, M., & Deljoo, A. (2012). A Wireless Sensor Network Solution for Precision Agriculture Based on Zigbee Technology. *Wireless Sensor Network*.

https://doi.org/10.4236/wsn.2012.41004

Khazaii, J. (2016). Agent-based modeling. *ASHRAE Journal*. https://doi.org/10.1007/978-3-642-24004-1_2

Lewis, F. L., Zhang, H., Hengster-Movric, K., & Das, A. (2014). Cooperative Control of Multi-Agent Systems: Optimal and Adaptive Design Approach. In *Journal of Chemical Information and Modeling*. https://doi.org/10.1007/978-1-4471-5574-4

Mardani, A., Jusoh, A., & Zavadskas, E. K. (2015). Fuzzy multiple criteria decision-making techniques and applications - Two decades review from 1994 to 2014. *Expert Systems with Applications*. https://doi.org/10.1016/j.eswa.2015.01.003

Medineckiene, M., Zavadskas, E. K., Björk, F., & Turskis, Z. (2015). Multi-criteria decision-making system for sustainable building assessment/certification. *Archives of Civil and Mechanical Engineering*. https://doi.org/10.1016/j.acme.2014.09.001

Report, P., & Sharma, A. (2019). *Water Quality Detection using Wireless Sensor Network Department of Computer Science Engineering and Information Technology Jaypee University of Information Technology Waknaghat , Solan-173234 , , Himachal Pradesh May 2019 Certificate*. (May).

United Nations. (2015). Transforming our world: the 2030 Agenda for Sustainable Development. United Nations Sustainable knowledge platform. *Sustainable Development Goals*. https://doi.org/https://sustainabledevelopment.un.org/post2015/transformingourworld

Wade, T. J., Calderon, R. L., Brenner, K. P., Sams, E., Beach, M., Haugland, R., … Dufour, A. P. (2008). High sensitivity of children to swimming-associated gastrointestinal illness: Results using a rapid assay of recreational water quality. *Epidemiology*. https://doi.org/10.1097/EDE.0b013e318169cc87

Wang, W., Chen, J., & Hong, T. (2018). Occupancy prediction through machine learning and data fusion of environmental sensing and Wi-Fi sensing in buildings. *Automation in Construction*. https://doi.org/10.1016/j.autcon.2018.07.007

Zhu, Y. L., & Xie, L. Q. (2013). Forest fire detection system based on zigbee wireless sensor network. *Advanced Materials Research*. https://doi.org/10.4028/www.scientific.net/AMR.694-697.961