
VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

STUDIJŲ PROGRAMA: INFORMATIKA

# Adaptyvių kompiuterinių sistemų formalus modeliavimas ir verifikavimas naudojant modelių patikrinimo įrankį UPPAAL

Formal modelling and verification of an adaptive computer-based system using UPPAAL model checker

Baigiamasis Magistro darbas

| | | |
|---|---|---|
| Atliko: | Daniel Daukševič | (parašas) |
| Vadovas: | prof. dr. Linas Laibinis | (parašas) |
| Recenzentas: | doc. dr. Haroldas Giedra | (parašas) |

Vilnius

2023

**Padėka**

Norėčiau nuoširdžiai padėkoti kiekvienam žmogui, prisidėjusiam prie šio darbo bei kitų mano akademinių pasiekimų Magistro studijų metu.

Pirmiausiai norėčiau padėkoti darbo vadovui, prof. dr. Linui Laibiniui, už jo laiką ir indėlį. Jo nuolatinė parama bei žinios buvo ypatingai svarbios šio darbo tyrimo metu.

Taip pat norėčiau padėkoti darbo recenzentui, doc. dr. Haroldui Giedrai, už vertingus patarimus ir taiklius pastebėjimus.

Pabaigai, norėčiau padėkoti savo šeimai bei kolegoms.

## Santrauka

Adaptyvios kompiuterinės sistemos – sudėtingos įrenginių struktūros, gebančios atpažinti bei atitinkamai prisitaikyti prie aplinkos pokyčių arba darbo trikdžių. Kompiuterinės sistemos yra formaliai verifikuojamos – jų korektiškumas ir teisingumas suformuluotų reikalavimų atžvilgiu įrodomas taikant formalius matematinius metodus. Šio darbo tikslas yra formaliai verifikuoti adaptyvią teritoriją valančią kompiuterinę robotų sistemą UPPAAL formalaus modeliavimo kalba ir asocijuotu įrankiu taikant modelių patikrinimo metodą, palyginti skirtingas sistemos konfigūracijas keičiant sistemos parametrus ir jos savybes bei tikslinant keliamus sistemai reikalavimus. Analizuojamą sistemą sudaro keturi robotų tipai, du vidiniai sistemos mechanizmai bei modeliuojantis gedimus procesas. Tyrime yra siūlomas hibridinis sistemos architektūros tipas, apjungiantis dvi plačiai naudojamas strategijas – hierarchinį ir saviorganizuojantį, apibrėžti taikomi sistemai reikalavimai. Eksperimentuose taikomas tiek klasikinis, tiek statistinis modelių patikrinimo metodas, besiremiantis simuliavimu bei tikimybiniu savybės tenkinimo įvertinimu. Tyrimo rezultatų pagrindu yra pateikiami sukaupti pastebėjimai ir rekomendacijos analizuojamos sistemos kūrimui ir tolimesniems tyrimams.

**Summary**

Adaptive computer systems are complex device structures capable of recognizing and accordingly adapting to environmental changes or work disturbances. Computer systems are formally verified; their correctness in relation to the formulated requirements is proven by applying formal mathematical methods. The purpose of this work is to formally verify the adaptive robot cleaning computer-based system using the UPPAAL formal modelling language and associated tool by applying the model checking method, comparing different system configurations by changing the system parameters and its properties, and specifying the system requirements. The analysed system consists of four types of robots, two internal mechanisms of the system, and a process simulating failures. The study proposes a hybrid type of system architecture that combines two widely used strategies, hierarchical and self-organizing, and defines the requirements applied to the system. In the experiments, both classic and statistical (based on simulation and probabilistic evaluation of property satisfaction) model checking methods are applied. Based on the research results, accumulated observations and recommendations for the development of the analysed system and further research are presented.

**Turinys**

# Įvadas

Kompiuterinės sistemos (ang. *computer-based systems*) – sudėtingos įrenginių struktūros, dažnu atveju sudarytos iš komponenčių – posistemių (ang. *subsystem*), kurių atliekami veiksmai remiasi kompiuterių vykdomais matematiniais skaičiavimais arba aplinkos stebėjimu (ang. *monitoring*) ir valdymu. Moderniais laikais kompiuterinės sistemos yra plačiai paplitusios – jos dalyvauja bemaž kiekvienoje žmogaus gyvenimo srityje. Kompiuterinės sistemos yra naudojamos siekiant optimizuoti arba palengvinti žmogaus darbą, apdoroti ir analizuoti informaciją apie įvairius pasaulio aspektus [ECP18, KrK88].

Dalies kompiuterinių sistemų – vadinamų kritinėmis (ang. *critical*) sistemomis – veiksmai gali turėti tiesioginę įtaką žmogaus materialinei arba fizinei gerovei. Tokių sistemų sertifikavimo metu jų atliekamų veiksmų (skaičiavimų) korektiškumui yra taikomi griežti reikalavimai – būtini įrodymai, jog sistema visada veikia korektiškai, o atliekami skaičiavimai atitinka numatytus apibrėžimus. Toks fundamentalus sistemos patikrinimas yra galimas formalių metodų pagalba [HiC10].

Formalūs metodai (ang. *formal methods*) – matematiniai metodai ir technikos, skirti programinės (ang. *software*) ir techninės (ang. *hardware*) įrangos specifikavimui, projektavimui ir verifikavimui. Formaliuose metoduose yra taikomi teoriniai įrankiai, pvz. logika, formalios kalbos, automatų teorija. Formalūs metodai yra naudingi tuo, jog leidžia išnagrinėti visą įmanomų sistemos būsenų aibę bei patikrinti, ar apibrėžta teisingumo (ang. *correctness*) arba saugumo (ang. *safety*) savybė yra patenkinama (teisinga) visose būsenose. Vienu plačiausiai naudojamų formalių metodų yra formalus verifikavimas [But16, Sch01, Sys20, WLB+09].

Formalus verifikavimas (ang. *formal verification*) – algoritmo (kompiuterinės programos arba sistemos) korektiškumo ir teisingumo suformuluotų reikalavimų atžvilgiu įrodymas taikant formalius matematinius metodus. Egzistuoja du pagrindiniai verifikavimo metodai – modelių patikrinimas ir teoremų įrodymas [Bje05, RaR20].

Baigiamojo magistro darbo metu buvo atliktas tyrimas, kuriame buvo formaliai verifikuojama adaptyvi kompiuterinė robotų sistema taikant modelių patikrinimo metodą. Tyrimą galima laikyti ankstesnėje studijų pakopoje atlikto baigiamojo darbo tyrimo pratęsimu. Sistema buvo modeliuojama kita formalaus modeliavimo kalba (UPPAAL), kuriant sistemos modelį buvo tyrinėjamos laiko savybės. Ankstesniame tyrime buvo skiriamas didelis dėmesys sistemos kokybinėms (ang. *qualitative*) savybėms – $TLA^+$ formalaus specifikavimo kalba sukurtame modelyje buvo tiriamas

invariantų, apibrėžiančių sistemos reikalavimus, tenkinimas – pvz. nepriklausomai nuo gedimų skaičiaus, sistema visada pilnai atlieka darbą, vienu metu sistemos komponentei gali būti priskirta tik viena užduotis ir t.t. Šiame darbe akcentuojamas laiko aspektas – tyrinėjamas sistemos reakcijos greitis į įprastas ir nenumatytas situacijas. Šis aspektas tiesiogiai lėmė modeliavimo kalbos ir įrankio (UPPAAL) pasirinkimą. Pasiekti darbo rezultatai apima sukurtus sistemos modelius ir formaliai patikrintas savybes, kurių pagrindu galima samprotauti apie sistemos ypatumus, potencialiai silpnas vietas, pakeitimų įtaką sistemos darbui, skirtumą tarp tos pačios sistemos konfigūracijų [Dau21].

Mokslo tiriamojo darbo temos pasirinkimas yra grindžiamas faktu, jog kiekviena kritinė kompiuterinė sistema privalo būti sėkmingai verifikuota – šio proceso aktualumas ir svarba yra nekintantys. Formalus verifikavimas yra suprantamas kaip esminė ir būtina kiekvieno didelės apimties ir svarbos kompiuterinės sistemos kūrimo projekto dalis; formalaus modeliavimo įgūdžiai srities profesionalų yra įvardijami kaip programuotojo kompetenciją bei gilesnį srities išmanymą suteikiančios žinios. Adaptyvios kompiuterinės sistemos yra suprantamos kaip vertingi ir naudingi įrenginiai, kurių autonomiškumo plėtimas kūrimo metu reikštų mažesnį resursų poreikį sistemos palaikymui eksploatavimo metu. Formalaus verifikavimo procesas, taikomas projektavimo metu siekiant rasti fundamentalias sistemos klaidas bei trūkumus, gali būti naudojamas kompiuterinių sistemų adaptyvumo bei autonomiškumo simuliavimui, siekiant išanalizuoti planuojamų pakeitimų įtaką sistemos darbui ir greitaveikai.

Darbo tikslas – verifikuoti adaptyvią kompiuterinę sistemą UPPAAL formalaus modeliavimo kalba ir asocijuotu įrankiu taikant modelių patikrinimo metodą. Darbo uždaviniai:

- Išanalizuoti mokslinę literatūrą, atliktus mokslinius tyrimus (ang. *state-of-the-art*) tiriant ir modeliuojant panašias arba artimas analizuojamai adaptyvias sistemas siekiant sudaryti teorinį pagrindą mokslo darbo tyrimui – palyginti egzistuojančius metodus ir aplinkas, identifikuoti tinkamiausius apibrėžtų savybių tyrimui;
- Išsiaiškinti esminius bei papildomus pasirinktos kompiuterinės sistemos reikalavimus bei verifikuotinas savybes (pvz. atsparumą gedimams, reakcijos greitį) siekiant sudaryti kuo pilnesnį sistemos įsivaizdavimą bei sukurti kuo tikslesnį sistemos modelį;
- Identifikuoti pasirinktos formaliam verifikavimui kompiuterinės robotų sistemos charakteristikas ir jos parametrus;
- Sukurti formalų sistemos modelį formalaus modeliavimo kalba UPPAAL;
- Atlikti tyrimą, kurio metu verifikuoti adaptyvią kompiuterinę sistemą naudojant UPPAAL

modelių patikrinimo įrankį, keičiant sistemos parametrus bei jos savybes, tikslinant keliamus sistemai reikalavimus, palyginti skirtingas sistemos konfigūracijas;

- Išanalizuoti gautus rezultatus bei jų pagrindu apibrėžti pakeitimų įtaką sistemai ir jos veikimui, pabrėžiant potencialias sistemos silpnąsias vietas;
- Pateikti darbo metu sukauptus pastebėjimus ir rekomendacijas verifikuojamos sistemos kūrimui ir tolimesniems tyrimams.

Baigiamąjį magistro darbą sudaro papildytos trys mokslo tiriamojo darbo dalys. Remiantis ankstesnių darbo dalių recenzijomis buvo sukonkretinti tikslas ir uždaviniai, patikslinti laukiami rezultatai ir atliktas detalus tyrimo palyginimas su kitų autorių eksperimentais modeliuojant ir verifikuojant teritoriją valančią robotų sistemą (VI skyrius).

# 1 Formalus modeliavimas ir verifikavimas

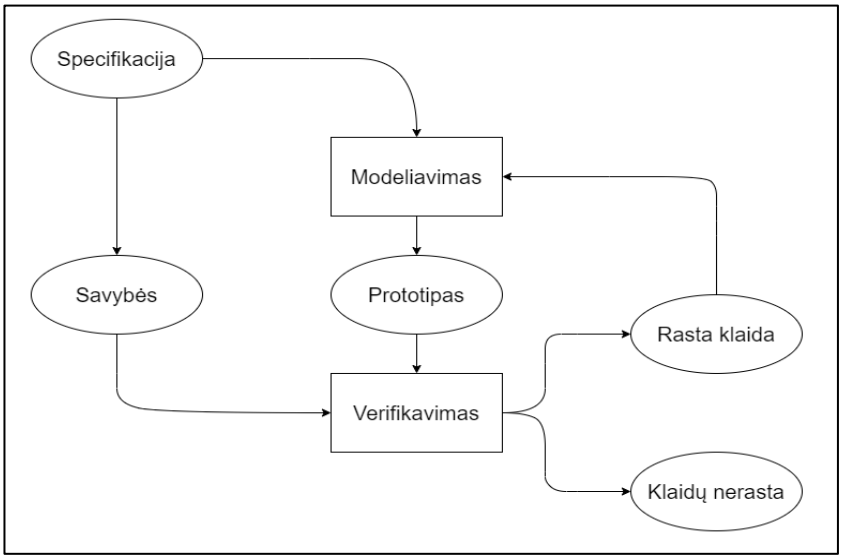
Formalus verifikavimas – procesas, kurio metu yra tiriama, ar nagrinėjamas objektas (algoritmas, kompiuterinė sistema) pasižymi atitinkamomis savybėmis, tenkina keliamus reikalavimus bei pilnai atlieka suformuluotus uždavinius. Pradedant nuo bendrų ir intuityviai suprantamų charakteristikų, pvz. aklaviečių (ang. *deadlock*) – situacijų, kai sistema dėl netenkinamų sąlygų negali atlikti jokio veiksmo – nebuvimo, iki unikalių, savitų būtent tiriamajam objektui atributų ir požymių, aprašytų specifikacijoje [Bje05].

Verifikavimo sąvokos nederėtų tapatinti su validacija, kadangi pastarasis terminas apibrėžia algoritmo darbo rezultatų tinkamumo (pagal vartotojo poreikius) tikrinimą. Verifikavimas yra atliekamas siekiant nustatyti paties algoritmo veikimo korektiškumą, t.y. ar pilnai yra išpildyti reikalavimai (specifikacija). Verifikavimo ir validacijos sąvokos, nors dažnai minimos ir naudojamos kartu, negali būti laikomos alternatyva viena kitai.

Tam, kad objektą galima būtų formaliai verifikuoti, būtina sukurti jo matematinį atitikmenį – teorinę (loginę, matematinę) struktūrą, pasižyminčią esminiais, būdingais tiriamajam objektui bruožais, vykdomų skaičiavimų ir sprendimų priėmimo specifika. Tokia struktūra yra formalus modelis – objekto atvaizdavimas, apibrėžtas taikant matematinius metodus. Bendru atveju formalus modelis yra abstraktus – itin svarbu identifikuoti pagrindines ir esmines savybes, keliamus reikalavimus, užduotis ir jų vykdymą. Informacija, kurios įtaka tyrimui (arba objekto veikimui bendrai) yra minimali, yra ignoruojama. Galima teigti, jog tokiu būdu yra sukuriamas objekto brėžinys – formali specifikacija arba aprašymas. Formalų modelį taip pat galima traktuoti kaip objekto juodraštį (ang. *blueprint*) – prieš faktinę realizaciją egzistuojantį objekto apibrėžimą, kurį galima papildyti bei keisti siekiant identifikuoti optimaliausią sprendimo architektūros versiją [Lam02].

Formalus verifikavimas – ilgas, ir dažnu atveju, iteratyvus procesas (žr. *Pav. 1*), kuris yra pradedamas nuo tiriamojo objekto analizės. Bendru atveju, kai yra verifikuojama konkreti sistema, teorinį analizės pagrindą sudaro specifikacija, t.y. sistemos aprašymas. Remiantis sukaupta informacija yra modeliuojamas sistemos prototipas, formalizuojamos norimos įrodyti savybės arba reikalavimai. Sukūrus modelį, atliekamas verifikavimas. Aptikus klaidą, grįžtama prie sistemos modeliavimo – peržiūrima modelio architektūra, konkrečių veiksmų patikslinimas – ir verifikuojama nauja modelio versija (konfigūracija). Iteracijos vyksta tol, kol nėra pašalintos visos klaidos, tačiau tai nebūtinai yra viso proceso pabaiga. Formalų verifikavimą galima tęsti – išanalizavus gautus

rezultatus, formalizavus papildomus reikalavimus arba siekiant patikrinti alternatyvų sprendimą [BaK08].




*Pav. 1. Formalaus verifikavimo proceso schema*

Formalizavus probleminę sritį (algoritmą, kompiuterinę sistemą) – transliavus jos esmines sudedamąsias dalis ir faktus bei atliekamų skaičiavimų apibrėžimus į matematinę kalbą – gaunamas formalus tos srities modelis (specifikacija), kurį galima verifikuoti bei analizuoti taikant:

- Teoremų įrodymą;
- Modelių patikrinimą arba statistinį modelių patikrinimą;
- Simuliavimą.

## 1.1 Teoremų įrodymas

Teoremų įrodymas (ang. *theorem proving*) – automatizuotas objekto kaip teiginių logikos formulės teisingumo įrodymas remiantis formaliu samprotavimu. Taikant šį formalaus verifikavimo metodą siekiama įrodyti, kad apibrėžianti tiriamą objektą formulė yra teisinga. Formalizuojant probleminę sritį yra sukuriama ją atitinkanti konkreti teorija, įtraukianti bazines sąvokas (apibrėžimus), prielaidas, apribojimus (predikatus) bei formaliai suformuluotas laukiamas sistemos savybes bei išvadas (teoremas). Verifikavimo proceso tikslas – rasti ryšį tarp apibrėžtų prielaidų ir išvadų, įrodyti jog toks samprotavimas yra pagrįstas. Įrodymo procesas (ang. *proving process*) gali būti tiek interaktyvus (žmogaus ir automatizuotos sistemos sąveikos pagalba, kai žmogus turi galimybę rinktis tarp galimų įrodymo žingsnių ir strategijų), tiek automatizuotas (loginius išvedimus atlieka kompiuteris remdamasis duota teorija ir laukiama išvada). Dažnai

verifikavimo procesas yra dviejų įrodymo būdų junginys, kai interaktyvūs ir pilnai automatizuoti įrodymo žingsniai keičia vienas kitą [Sch01].

Šis formalaus verifikavimo metodas reikalauja gilaus analizuojamo objekto (problemos, kurią objektas turi išspręsti) supratimo bei matematinės logikos išmanymo. Lyginant su kitu formalaus verifikavimo metodu, modelių patikrinimu, teoremų įrodymas reikalauja žymiai daugiau resursų ir laiko. Įrodymo paieškos procesas trukmės atžvilgiu nėra ribojamas, todėl dažnu atveju užima kelias dienas, savaites ar net mėnesius. Teoriškai, formaliai verifikuojamas objektas (pvz. kompiuterinė sistema) gali būti toks sudėtingas, kad atsakymo paieška truks neapibrėžtą laikotarpį. Praktikoje tai lemia dažnai negatyvų požiūrį į šį formalaus verifikavimo metodą – vartotojo atžvilgiu teoremų įrodymas yra neefektyvus, kadangi potencialiai užima per daug laiko. Nepaisant ilgai trunkančio įrodymo proceso, jei teoremų įrodymas grąžina vienareikšmišką atsakymą, moksliniu požiūriu, jis yra pilnas ir neginčijamas [Rus01].

Esti įvairių teoremų įrodymo metodą palaikančių įrankių, besiremiančių skirtingomis logikomis ir įrodymo algoritmais. Dažniausiai naudojami interaktyvūs teoremų įrodytojai (ang. *interactive theorem prover* arba *proof assistant*):

- VDM – formalus metodas:
  - VDM++ – objektiškai orientuotoms ir lygiagrečioms sistemoms;
- B metodu (ang. *B-method*) besiremiantys įrodytojai paskirstytoms ar reaktyviosioms sistemoms [ALN+05]:
  - Atelier-B;
  - Rodin;
- Daug žinių ir ekspertizės reikalaujantys, bendro pobūdžio (ang. *general purpose*) galingi teoremų įrodytojai [NML+19]:
  - Isabelle;
  - Coq;
  - HOL;
  - PVS;
  - Metamath;
  - Mizar;
  - ProofProver.

## 1.2 Modelių patikrinimas

Modelių patikrinimas (ang. *model checking*) – automatizuota tiriamo objekto veikimo analizė būsenų perrinkimo ir patikrinimo būdu, atliekama siekiant nustatyti ar yra tenkinami apibrėžti reikalavimai, o darbas yra atliekamas korektiškai (pvz. nėra galimybės patekti į kritinę būseną). C.Baier ir J.-P.Katoen knygoje „Principle of Model Checking“ yra apibrėžiamas modelių patikrinimo metodo principas – „*Modelių patikrinimas tai automatizuota technika, kuri, pateikus baigtinį sistemos modelį ir formalią savybę, sistemingai tikrina ar ta savybė galioja (tam tikroje būsenoje) tam modeliui*“ [BaK08].

Skirtingai nuo teoremų įrodymo metodo, modelių patikrinimas yra pilnai automatizuotas bei greitas – neretai atsakymas yra pateikiamas per kelias minutes (skaičiavimo trukmė priklauso nuo analizuojamo objekto bei įrodomų savybių sudėtingumo). Modelių patikrinimą galima vadinti „informatyvesniu“ atsižvelgiant į tai, kiek informacijos apie verifikuojamą sistemą galima gauti tyrimo metu. Net jeigu patikrinimas buvo nutrauktas identifikavus klaidą, modelių patikrinimas vietoje pranešimo apie nepasisekusį tyrimą suteikia galimybę panagrinėti modelį identifikuojant sistemos klaidos situaciją. Jeigu patikrinimo metu buvo rasta būsena, sukelianti tiriamos savybės netenkinimą, modelių patikrintojas pateikia kontrpavyzdį (ang. *counterexample*), indikuojantį kaip modelis pasiekia klaidos būseną. Kontrpavyzdys – tai vykdymo (vieno iš tiriamojo objekto veikimo scenarijų) aprašymas nuo pradinės būsenos iki būsenos, kurioje verifikuojama savybė yra netenkinama. Simuliatoriaus (modelio patikrintojo dalies) pagalba kontrpavyzdį galima pažingsniui atkartoti, derinant (ang. *debugging*) ir analizuojant pokyčius [Deb18].

Modelių patikrinimą galima apibrėžti trimis etapais:

1. Modeliavimas – matematinio modelio kūrimas. Formalaus modeliavimo kalba yra sukuriamas tiriamojo objekto aprašymas, kuriame nurodomi sistemos veiksmai ir savybės. Atliekami greiti patikrinimai (simuliavimai) siekiant įsitikinti implementacijos tikslumu;
2. Patikrinimas – naudojant asocijuotą įrankį yra atliekama nuodugni automatizuota pateikto modelio analizė. Generuojant visus įmanomus sistemos veikimo scenarijus – perėjimus tarp būsenų – yra tikrinama ar suformuluoti reikalavimai yra tenkinami;
3. Analizė - nagrinėjami gauti rezultatai. Jei savybė yra patenkinama, ji yra griežtinama (jei įmanoma) arba formuluojama kita. Modelis yra tobulinamas, detalizuojamas – yra mažinamas abstrakcijos lygis. Nuosekliai tobulinamos modelio versijos yra susietos

formaliais tikslinimo (ang. *refinement*) ryšiais – tai leidžia tolimesniems patikrinimams remtis ankstesniųjų rezultatais. Jeigu buvo rastas savybės nepatenkinimo faktas, analizuojamas pateiktas kontrpavyzdys, o modelis ar pati savybė yra tikslinami – verifikuojama pakartotinai. Jei patikrinimas nutrūko dėl atminties stokos yra didinamas modelio abstrakcijos lygis, abstrahuojant neturinčias pastebimos ir tiesioginės įtakos verifikuojamai savybei objekto dalis [CGK+18, Mer00].

Vieni plačiausiai naudojamų modelių tikrintojų:

- NuSMV – dvejetainių sprendimo diagramomis (ang. *Binary Decision Diagram, BDD*) besiremiantis įrankis skirtas pakartotinio naudojimo komponenčių (ang. *reusable components*) sistemų verifikavimui;
- PRISM – tinkamas tikimybinių modelių (ang. *probabilistic models*) ir paskirstyto skaičiavimo sistemų patikrinimui;
- ProB – B metodu besiremiantis įrodytojas naudojamas kritinių saugumo sistemų verifikavimui;
- TLA$^+$ (TLC) – naudojamas lygiagrečių skaičiavimų sistemų modeliavimui ir patikrinimui [Lam02];
- SPIN – PROMELA modeliavimo kalba aprašytų modelių tikrintojas;
- ROMEO – realaus laiko (ang. *real-time* arba *reactive*) sistemų, modeliuojamų kaip Petri tinklai, modeliavimui, validacijai ir verifikavimui;
- DREAM – atviro kodo įrankis paskirstytoms realaus laiko įterptinėms (ang. *embedded, DRE*) sistemoms;
- UPPAAL – realaus laiko ir paskirstyto skaičiavimo sistemų formaliam modeliavimui ir verifikavimui.

## 1.3 Statistinis modelių patikrinimas

Modelių patikrinimo metodas netinka visoms kompiuterinėms sistemoms. Jei sistema pasižymi dideliu veikimo scenarijų skaičiumi, galimas „būsenų sprogimas“ (ang. *state explosion*) – dėl per didelio patikrinimų skaičiaus ir tuo sukelto laiko arba atminties trūkio modelių patikrinimas gali nesuformuluoti atsakymo visiškai. Siekiant išspręsti šią problemą buvo sukurtas hibridinis metodas – statistinis modelių patikrinimas. Vietoje nuoseklaus ir išsamaus kiekvienos galimos būsenos ir jų sekos patikrinimo yra atliekami objekto elgsenos simuliavimai, kaupiant rezultatus ir statistiką (pvz. Monte Karlo simuliavimą). Surinktos informacijos pagrindu

yra apibrėžiamos tikimybės jog objektas pasižymi pasirinkta savybe arba tenkina reikalavimą (gali pasiekti atitinkamą būseną). Skirtingai nuo klasikinio modelių patikrinimo, statistika besiremiantis metodas negali pateikti vienareikšmiškai tikslaus rezultato – antra vertus, esti galimybė apibrėžti klaidos tikimybę. Nepaisant to, statistinis modelių patikrinimas yra pripažįstamas kaip patikimas tyrimų įrankis [BDL+12a, DLL+15].

Statistinis modelių patikrinimas leidžia verifikuoti sudėtingesnes sistemas ir savybes. Verta pažymėti, jog šis pranašumas egzistuoja įsitikinimo sistemos korektiškumu kaina – vietoje pilno įrodymo yra suformuluojamas tikimybinis reikiamo sistemos veikimo įvertinimas. Nepaisant to, statistinis modelių patikrinimas yra plačiai naudojamas įvairiuose tyrimų srityse, pvz. sistemų biologijoje, automobilių ir aviacijos elektronikoje ir programinės įrangos inžinerijoje [Ver17].

Vieniems populiariausių statistinio modelių patikrinimo įrankių yra priskiriami:

- VeStA (bei jo plėtinys MultiVeStA) – Java kalba parašytas statistinės analizės įrankis tikimybinių sistemų verifikavimui;
- APMC (Approximate Probabilistic Model Checker) – Monte Karlo metodą implementuojantis modelių patikrintojas, naudojamas įvertinti tikimybės sistemos pasirinktos savybės patenkinimą;
- UPPAAL SMC – UPPAAL plėtinys stochastinių hibridinių sistemų modeliavimui ir verifikavimui;
- COSMOS – implementuotas C++ kalba statistinio verifikavimo įrankis;
- GreatSPN – įrankis, skirtas kiekybinei ir kokybinei paskirstytų sistemų analizei;
- MRMC (Markov Reward Model Checker) – C kalba parašytas komandinės eilutės įrankis. Nors šis įrankis yra tikimybinis modelių tikrintojas, naudojantis jame implementuotomis Monte Carlo metodu besiremiančiomis technikomis galima atlikti statistinę analizę [PMT20].

## 1.4 Simuliavimu besiremiantis verifikavimas

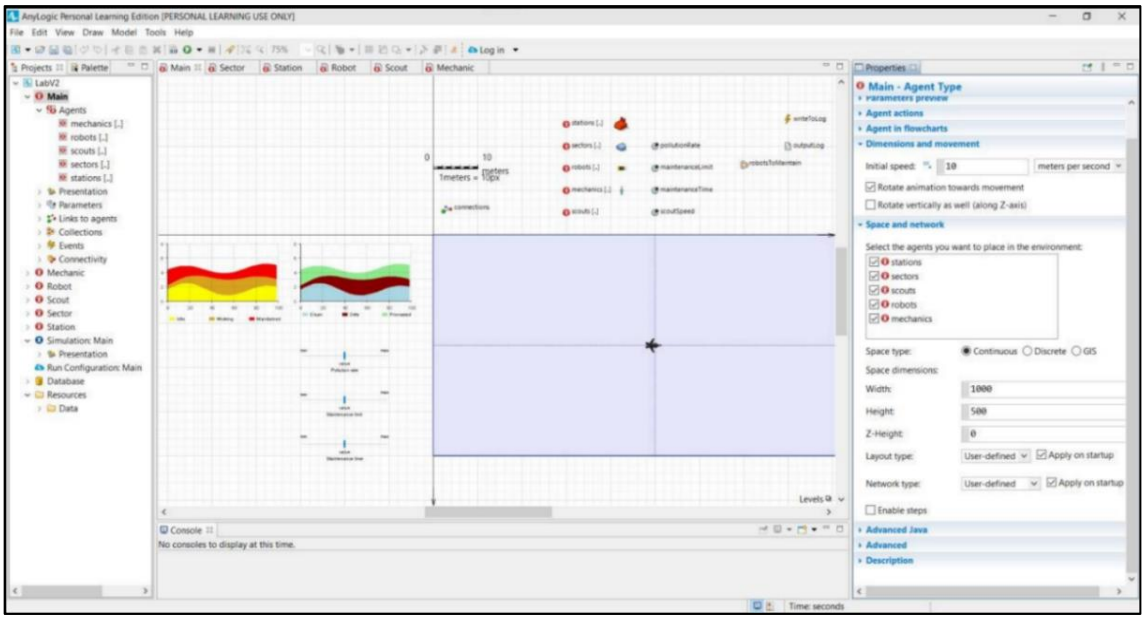
Simuliavimas – realaus pasaulio procesų imitacija. Vykdomas sistemos prototipas (modelis) atvaizduoja pagrindines pasirinktos sistemos (arba proceso) charakteristikas, o simuliavimas – modelio kaitą iš laiko perspektyvos. Simuliavimu besiremiantys metodai dažnu atveju, lyginant su pagrindiniais formalaus verifikavimo metodais, reikalauja pastebimai mažiau atminties ir laiko resursų. Simuliuoti tiriamo objekto darbą galima tiek siekiant patikrinti, ar modeliuojama architektūra (konfigūracija) atitinka keliamus darbo tikslus bei reikalavimus, tiek

tiriant pasirinkto parametro arba atsitiktinės reikšmės įtaką objekto darbui ir pasiekiamiems rezultatams [Her90].

Kadangi simuliavimas, skirtingai nuo dviejų pagrindinių formalaus verifikavimo metodų, neatlieka fundamentalaus tiriamojo objekto patikrinimo (nepriklausomai nuo bandymų skaičiaus, nėra užtikrinama jog buvo patikrinti visi įmanomi darbo vykdymo scenarijai), jo atlikimo metu gauti rezultatai nėra vertinami kaip vienareikšmiškai ir pilnai atitinkantys realybę. Statistinis modelių patikrinimas šiuo atžvilgiu yra laikomas patikimesniu įrankiu. Antra vertus, dėl pastebimai mažesnių reikalavimų laiko ir resursų prasme, simuliavimas yra plačiai naudojamas praktikoje, skirtingose įvairaus dydžio įmonėse, siekiant atrasti konceptualias architektūros arba implementacijos klaidas kuriamuose sprendimuose. Tarp plačiausiai paplitusių simuliavimo įrankių yra:

- Simulink – MATLAB pagrindu sukurta grafinė programavimo aplinka dinaminių daugiadomenių (ang. *multidomain*) sistemų simuliavimui;
- Dymola – Modelica modeliavimo kalba besiremiantis įrankis skirtas simuliuoti integruotas ir sudėtingas sistemas (naudojamas automobilių, robotikos, kosmonautikos ir kt. srityse);
- SimulationX – techninių sistemų simuliavimas;
- Wolfram SystemModeler – kiber-fizinių sistemų modeliavimo ir simuliavimo aplinka;
- SimPy – Python karkasas (ang. *framework*) skirtas diskrečiųjų įvykių modeliavimui.

Vienas iš plačiausiai pramonėje naudojamų įrankių yra AnyLogic (žr. *Pav. 2*) [Any22].



*Pav. 2. AnyLogic grafinis editorius*

Šiame įrankyje yra integruoti keli simuliavimo modeliavimo (ang. *simulation modelling*) metodai:

- Sistemos dinamika – nedeterministinio sudėtingos sistemos elgesio tyrimas;
- Diskretieji įvykiai – sistemos kaip diskrečios įvykių sekos modeliavimas;
- Agentų modeliavimas (ang. *agent-based modelling*) – autonominių agentų (kaip pavienių individų, taip ir grupių, organizacijų) veiksmų formalizavimas.

AnyLogic taip pat galima kurti daugiametodinius (ang. *multi-method*) modelius. Įrankyje yra sukurtos bibliotekos, skirtos konkrečioms pramonės šakoms, pvz.: geležinkelių ar kelių eismo, gamybai, sandėliavimui ir t.t., integruoti realaus pasaulio žemėlapiai. Modelius simuliavimo metu galima atvaizduoti tiek dvimatės, tiek trimatės grafikos pavidalu [Gri21].

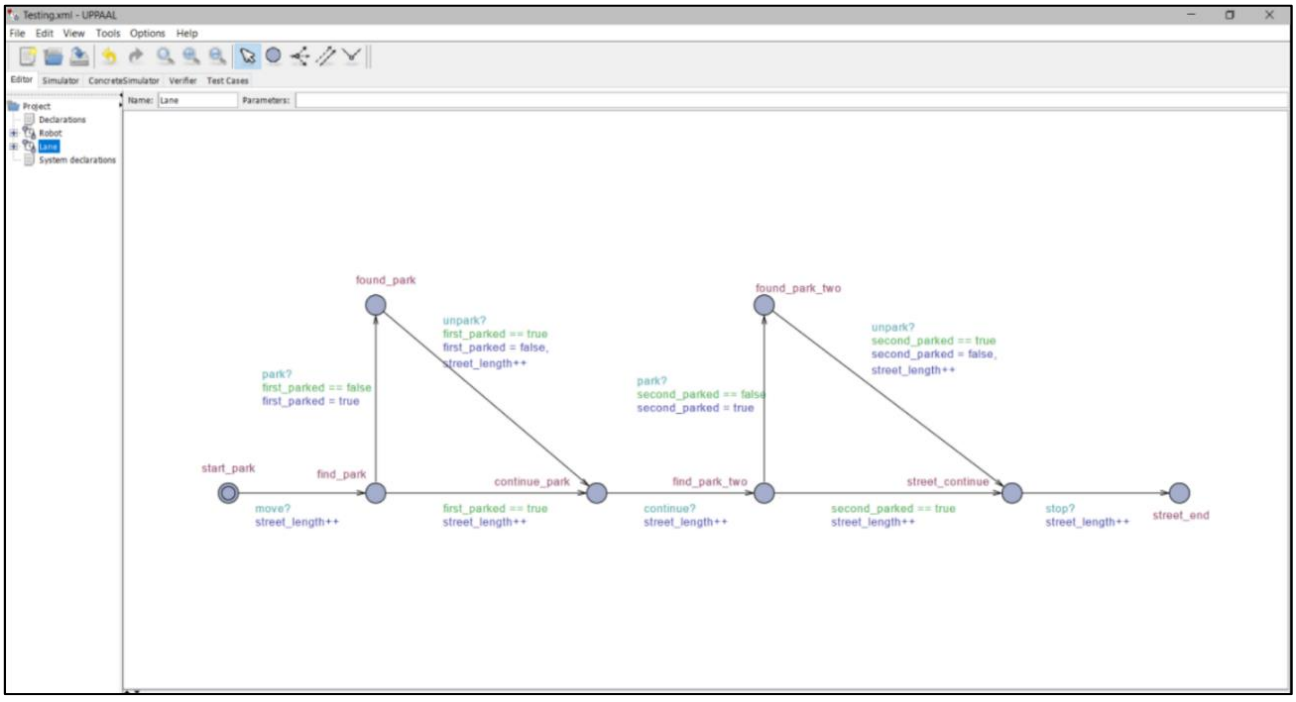
Atsižvelgiant į pasirinktos tyrimui kompiuterinės sistemos specifiką (darbo paskirstymas, autonomiškumas) bei siekiamas analizuoti charakteristikas (laiko savybės) formaliam modeliavimui ir verifikavimui buvo pasirinktas UPPAAL įrankio plėtinys UPPAAL SMC, implementuojantis statistinio modelių patikrinimo metodą. Lygiagrečiai, papildant pagrindinį tyrimą, kuriamas ir simuliuojamas analogiškas sistemos prototipas naudojant AnyLogic.

# 2 UPPAAL

Šio darbo tyrimui atlikti buvo pasirinkta UPPAAL integruota įrankių aplinka (ang. *integrated tool environment*) skirta realaus laiko (ang. *real-time*) sistemų formaliam modeliavimui, validacijai ir verifikavimui. Šis įrankis yra pritaikytas sistemoms, kurias galima modeliuoti kaip nedeterministinį procesų rinkinį su baigtine valdymo struktūra (ang. *finite control structure*), kurio komponentės komunikuoja tarpusavyje naudodamos kanalus ir (arba) bendras duomenų struktūras, kintamuosius. Įprastai UPPAAL yra naudojamas komunikavimo protokolų ir sistemų, kuriose laiko aspektas yra ypatingai svarbus, verifikavimui [BLL+96, Upp19].

UPPAAL įrankį (žr. *Pav. 3*) sudaro trys pagrindinės komponentės:

- Apibrėžimo (ang. *description*) kalba – analizuojamo objekto veiksmai (darbas) yra modeliuojami kaip automatų tinklas;
- Simulatorius – įrankis, leidžiantis patikrinti įmanomus dinaminius sistemos vykdymus;
- Verifikatorius – įrankis, skritas dinaminio verifikuojamo objekto elgesio analizei, leidžiantis verifikuoti invariantus ir *liveness* (arba *reachability*) sistemos savybes.




*Pav. 3. UPPAAL grafinis redaktorius*

## 2.1 Teoriniai pagrindai

Bendru atveju, samprotaujant apie pasiekiamumo (ang. *reachability*) savybę, yra įvedama laiko trukmės sąvoka – pvz., „įvykus kritinei klaidai, per nurodytą laiko tarpą T sistema privalo pasiekti būseną X“. Siekiant modeliuoti tokio tipo savybes, būtina įvesti realaus laiko sąvoką. Kadangi UPPAAL įrankis buvo specialiai kuriamas realaus laiko sistemų verifikavimui, modeliuojant fundamentalia ypatybe tampa laikmačiai.

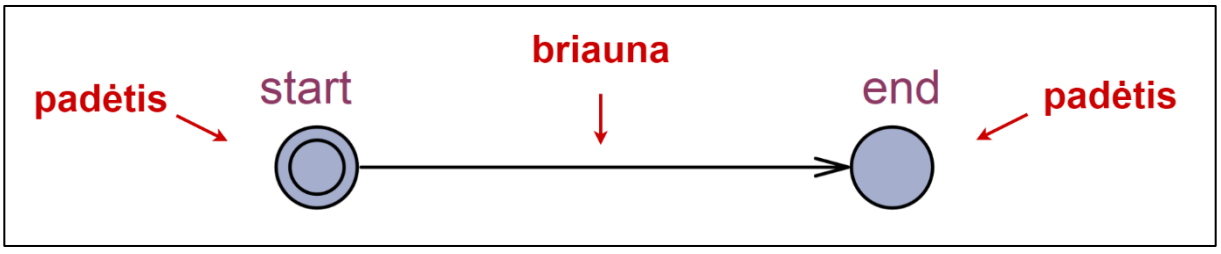
Laikmatis (ang. *clock*) – realaus skaičiaus tipo kintamieji skirti laiko pokyčio matavimui. Visi laikmačiai veikia tuo pačiu greičiu – taip yra sukuriamas globalus laiko pokytis. Reali laikmačio reikšmė gali būti *tested* arba *reset*.

Teorinis pagrindas, dažnai naudojamas realaus laiko sistemų modeliavimui ir verifikavimui, yra laiko automatų teorija. Laiko automatas (ang. *timed automata*) – baigtinis elementų rinkinys (L, l0, C, A, E, I), kur:

- L yra padėčių (ang. *locations*) aibė;
- l0 yra pradinė padėtis;
- C yra laikmačių (ang. *clocks*) aibė;
- A yra veiksmų, bendrų veiksmų (ang. *co-actions*) ir vidinio τ-veiksmo aibė;
- $E \subseteq L \times A \times B(C) \times 2C \times L$ yra aibė briaunų (t. y., perėjimų) tarp padėčių su tokių perėjimų sąlygomis ir susietais diskrečių kintamųjų keitimais, ir aibe atsistatančių laikmačių;
- $I : L \rightarrow B(C)$ invariantų priskyrimas atskiroms padėtims [BDL04].

## 2.2 Modeliavimo kalba

UPPAAL modelis yra lygiagrečiai vykstančių procesų aibė. Kiekvienas procesas yra laiko automatas, grafiškai vaizduojamas kaip grafas, kurio viršūnės yra lokacijos su jas jungiančiomis briaunomis (žr. *Pav. 4*).




*Pav. 4. Proceso modeliavimas UPPAAL*

Briaunoms yra priskiriami:

- Saugai (ang. *guard*) – loginės formulės, sudarytos iš kintamųjų ir laikmačių. Kai formulė yra teisinga, yra galimas perėjimas briauna. Vienu metu gali būti galimi perėjimai keliomis briaunomis – tokiu būdu atsiranda nedeterminuotumas;
- Atnaujinimai (ang. *updates*) – išraiškos (operacijos), kurios yra atliekamos modelio kintamiesiems, kai yra pereinama per briauną;
- Sinchronizacijos – veiksmai, atliekami siekiant koordinuoti du (arba daugiau) procesus;
- Pasirinkimai (ang. *selections*) – nedeterministinis reikšmių iš nurodyto intervalo arba aibės priskyrimas atitinkamiems kintamiesiems [Upp21].

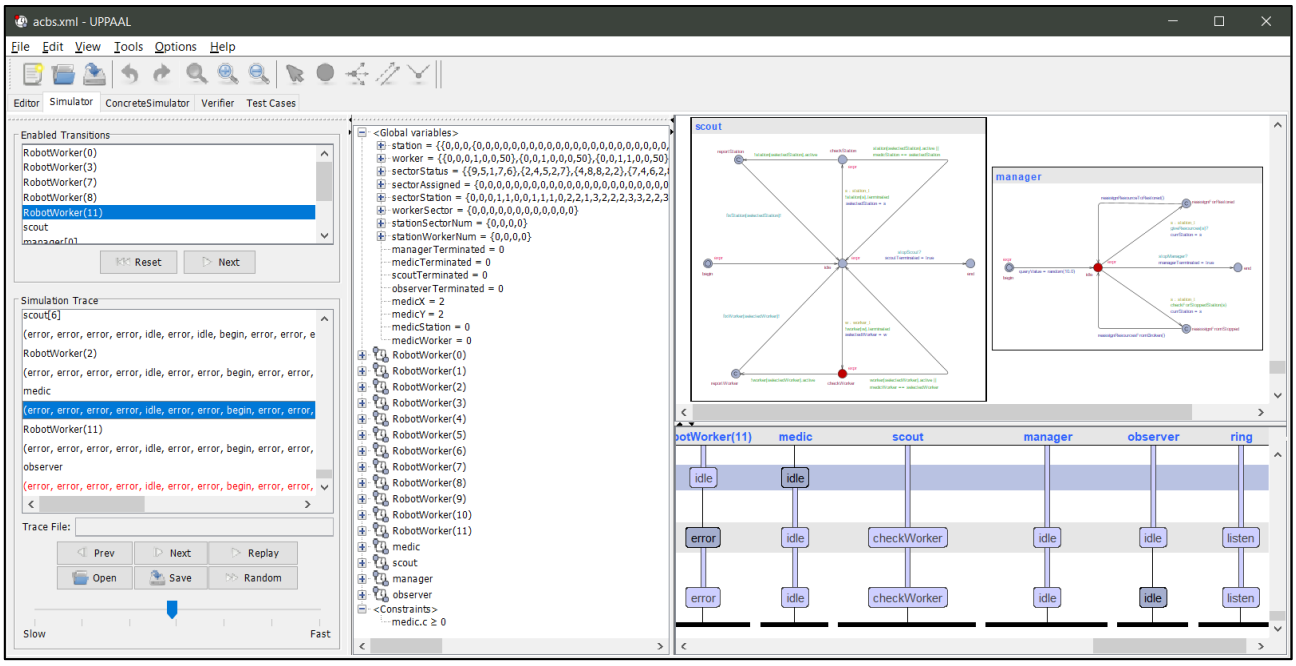
UPPAAL modeliavimo kalba taip pat yra išplėsta papildomais modeliavimo elementais ir galimybėmis:

- Šablonais (ang. *template*) – parametrizuotais laiko automatais atitinkančiais sistemos komponentus arba procesus;
- Konstantomis – nekeičiamomis po inicializavimo sveikojo skaičiaus tipo reikšmėmis;
- Ribotų sveikųjų skaičių kintamaisiais (ang. *bounded integer variables*), deklaruojamais kaip int[*min, max*], kur *min* ir max yra atitinkamai minimali ir maksimali reikšmės. Jei rėžiai nėra nurodyti, pagal nutylėjimą yra nurodomos reikšmės -2 15 ir 2 15;
- Dvejetainio sinchronizavimo kanalais, deklaruojamais *chan c*;
- Transliavimo kanalais, deklaruojamais *broadcast chan c*. Transliavimo sinchronizavimo metu vienas siuntėjas (*c!*) gali sinchronizuoti turimas žinias su atsitiktiniu gavėjų (*c?*) skaičiumi. Jei esama būsena gavėjui leidžia dalyvauti sinchronizavime, jis būtinai dalyvauja. Transliavimas niekada nėra blokuojamas, todėl net jei nėra nei vieno gavėjo, siuntėjas vis tiek išsiunčia pranešimą;
- Skubaus sinchronizavimo kanalais – jei tokiame kanale yra atliekama sinchronizacija, vėlavimai (ang. *delay*) negalimi;
- Masyvais laikmačiams, kanalams, konstantoms ir sveikojo skaičiaus tipo kintamiesiems. Deklaruojami nurodant jų dydį, pvz. *chan c[7]*; *clock c[4], int[1,6], u[8]*.

## 2.3 Simuliatorius

Simuliatorius (žr. *Pav. 5*) – validacijos įrankis, kurio pagalba galima interaktyviai tyrinėti galimus dinaminius sistemos vykdymus. Simuliatorius, kuris taip pat leidžia vizualizuoti verifikatoriaus sugeneruotus vykdymus, yra sudarytas iš keturių langų:

- Simuliavimo kontrolės – simuliavimo proceso valdymui;
- Kintamųjų – reikšmių kaitos sekimui ir analizei;
- Procesų – vykdomų operacijų sekimui ir analizei;
- Pranešimų sekos diagramų.




*Pav. 5. UPPAAL simuliatorius*

## 2.4 Verifikatorius

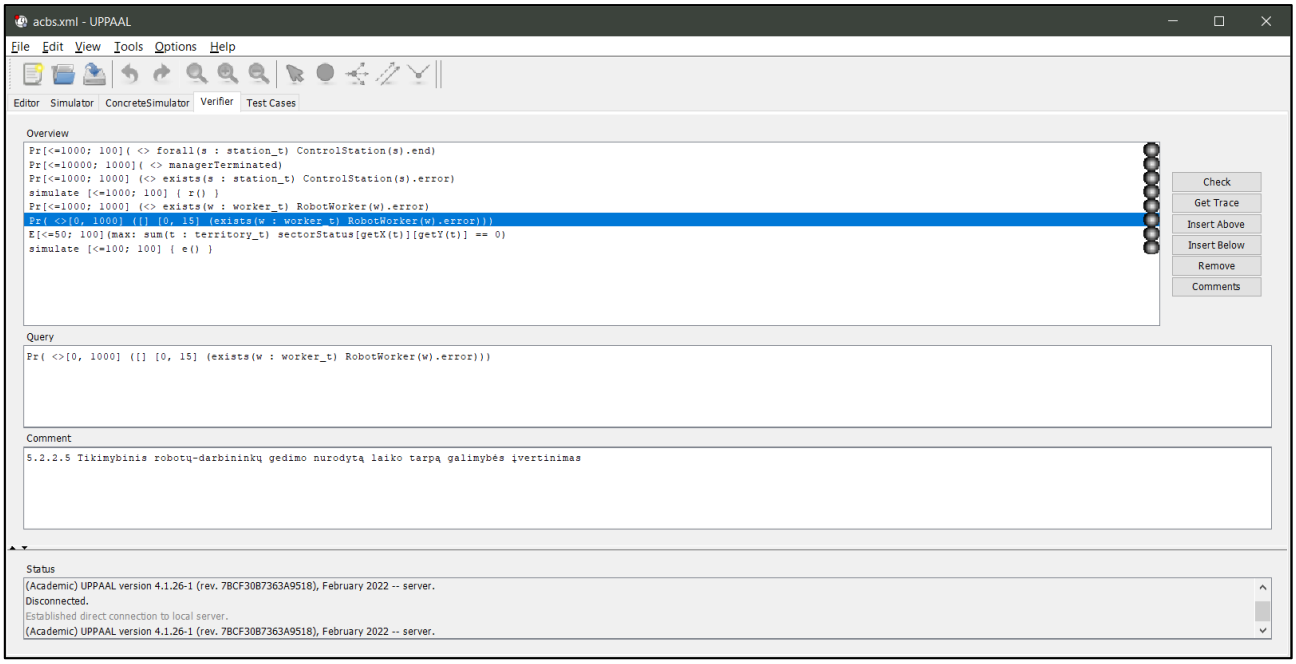
Verifikatorius (žr. *Pav. 6*) – įrankis, kurio pagalba galima tyrinėti užklausų (ang. *query*) kalba – skaičiavimo medžio logikos (ang. *computational tree logic*, *CTL*) poaibiu – aprašytas tiriamojo objekto savybes [BDL04]:

- Pasiekiamumo:
  - E ◊ p – „galiausiai (ang. *eventually*) p bus tiesa (būsena pasiekta)“;
- Saugumo – invariantų pavidalų apibrėžtos taisyklės, kurios visada turi būti teisingos (tenkinamos):
  - E □ p – „globaliai egzistuoja p“, tarp vykdymo kelių (ang. *execution path*) egzistuoja bent vienas kelias, kurio kiekvienoje būsenoje yra patenkinamas p;
  - A □ p – „globaliai visada yra p“, kiekviename vykdymo kelyje, kiekvienoje būsenoje yra tenkinamas p;
- Gyvybingumo (ang. *liveness*) – apibūdinama kaip „kažkas galiausiai bus pasiekta“:

- A ◊ p – „visada galiausiai p“ – kiekviename vykdymo kelyje, bent vienoje būsenoje, yra tenkinamas p;
- q ◊ p – „iš q seka p“ – kiekviename kelyje, kuriame yra q, vėliau bus tenkinamas p;

▪ Aklaviečių (ne)būvimo:

- E ◊ deadlock – „egzistuoja aklavietė“;
- A □ not deadlock – „nėra aklavietės“




*Pav. 6. UPPAAL verifikatorius*

## 2.5 UPPAAL SMC

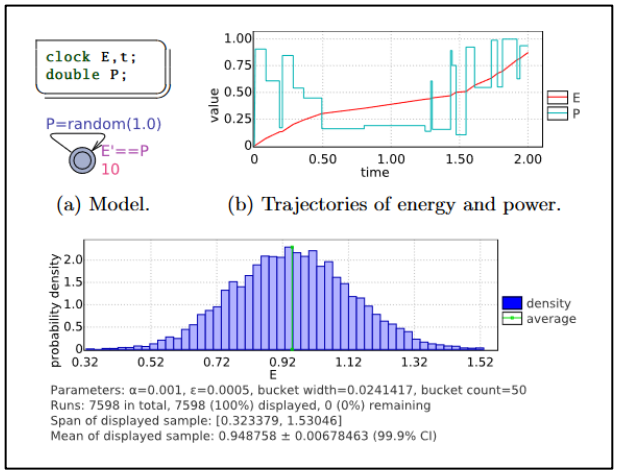
UPPAAL SMC – UPPAAL įrankio plėtinys (ang. *extension*), skirtas statistinio modelių patikrinimo metodo taikymui. Plėtinyje naudojamas formalaus modeliavimo formalizmas yra pagrįstas laiko automato (ang. *timed automata*) formalizmo stochastine interpretacija ir išplėtimu – tai leidžia modeliuoti tiriamojo objekto dinaminį elgesį (ang. *dynamical behavior*), tikrinti savybes jas įvertinant tikimybine verte.

Skirtingai nuo klasikinio modelių patikrinimo metodo implementacijos, kuriame buvo gaunami „taip-ne“ tipo rezultatai (ar reikalavimai yra tenkinami, ar visada yra pasižymima pasirinkta savybe) UPPAAL SMC rezultatus galima vizualizuoti, atvaizduoti duomenų grafikų (histogramų) pavidalu (žr. *Pav. 7*). Tai galima pasiekti papildytos užklausų kalbos dėka – pvz. grafikas (b) atvaizduoja simuliavimo, sugeneruoto užklausos

**simulate N [<=bound] {E1, ..., Ek}**

dėka. Šioje užklausoje:

- *N* žymi atliekamų simuliavimų skaičių;
- *bound* – simuliavimams skirto laiko viršutinį rėžį (ties kuriuo laiko vienetu turi būti sustabdytas eksperimentas;
- *E1, ..., Ek – k* išraiškų (ang. *expression*), kurios yra analizuojamos simuliavimų metu ir vizualizuojamos grafike [BDL+12b].




*Pav. 7. UPPAAL SMC generuojami grafikai ir histogramos [DLL+15]*

Tikimybinį įvertinimą – tikimybę, jog per apibrėžtą laiko tarpą išraiška bus įgyvendinta (nurodyta būsena bus pasiekta) – galima apskaičiuoti naudojant užklausą:

**Pr [bound] (<> E)**

Skaičiavimus galima praplėsti – jei pati tikimybė nėra aktuali, bet siekiama sužinoti ar ji patenka į intervalą, užklausą galima papildyti rėžiais, pvz. nurodant ar tikimybė yra didesnė už nurodytą reikšmę (*P0*):

**Pr [bound] (<> E) >= P0**

Arba lyginant tarpusavyje dvi tikimybes, pvz.:

**Pr [bound1] ( E1 ) >= Pr [bound2] ( E2 )**

Tikimybę, jog procesas kažkuriuo metu (tarp *min* ir *max*) nurodytą laiko tarpą (*e_bound*) bus

pasirinktoje būsenoje, galima taikant formulę:

**Pr( <>[min, max] ([][e_bound] E) )**

Šioje formulėje:

- *min* – laiko intervalo, kuriame yra tikrinama išraiška, pradžia;
- *max* – laiko intervalo, kuriame yra tikrinama išraiška, pabaiga;
- *e_bound* – išraiškos trukmės laiko rėžiai;
- *E* – išraiška; bendru atveju nurodoma kaip Procesas.Būsena.

Klasikinius eksperimentus UPPAAL SMC taip pat papildo numatomų reikšmių (ang. *expected value*) maksimumo:

**E[bound; N] (max : expr)**

ir minimumo:

**E[bound; N] (min : expr)**

įvertinimais.

Tiriant tikimybines savybes verifikuojamą modelį reikia pritaikyti statistinei analizei. Tam, kad UPPAAL SMC galėtų įvertinti modelio savybių ar reikalavimų patenkinimą, visi sinchronizavimo kanalai privalo būti transliavimo (*broadcast*) tipo [DLL+15].

Padėtims, kurios nėra įpareigotos (ang. *comitted*) arba apibrėžtos invariantais, turi būti nurodytas eksponentinis rodiklis (ang. *rate of exponential*). Remiantis eksponentiniu rodikliu, dažniausiai apibrėžiamu kaip dviejų sveikųjų skaičių santykis (pvz. 4:3), yra apskaičiuojama tikimybė jog procesas (esant galimybei pereiti į kitą būseną) paliks dabartinę būseną:

$$\mathbf{Pr = 1 - e^{-\lambda t}}$$

Šioje formulėje:

- $t$ – praėjęs laikas;
- $\lambda$ – eksponentinis rodiklis.

Uždelsimas prieš bet kurios operacijos atlikimą yra traktuojamas kaip realistiškas reiškinys. Visoms neįpareigotoms padėtims eksponentinis rodiklis bus nurodytas parametro forma – keičiant kintamojo reikšmę bus tyrinėjama įtaka sistemos veikimo laikui bei stabilumui.

# 3 Adaptyvios kompiuterinės sistemos

Moderniais laikais plačiai yra paplitusios kompiuterinės sistemos, gebančios atpažinti bei atitinkamai prisitaikyti prie aplinkos pokyčių arba darbo trikdžių. Tokiose sistemose, vadinamomis adaptyviomis (ang. *adaptive computer-based system*), galima išskirti lygiagrečiai vykdomas dvi veiklas – patį sistemos darbą (tikslo, kuriam sistema buvo sukurta ir eksploatuojama, siekimą) bei situacijos sekimą ir atitinkamų veiksmų siekiant niveliuoti pokyčių potencialią įtaką atlikimą. Sistemos, kuriose nėra implementuotas situacijos analizės mechanizmas, pasikeitus aplinkos sąlygoms (pvz. padidėjus oro temperatūrai, sustiprėjus vėjui) tampa pažeidžiamos, jų darbo kokybė prastėja arba yra nutraukiama. Sukurti universalią, prisitaikančią prie bet kokio tipo aplinkos poveikio, sistemą šiuo metu neatrodo įmanoma, tačiau aibės išorinių faktorių, kurių įtaka būtų atpažįstama ir atitinkamai apdorojama neprastėjant sistemos produktyvumui, praplėtimas yra kiekvienos kompiuterinės sistemos kūrėjų tikslas.

Formalus modeliavimas ir verifikavimas reikalauja tam tikro abstrakcijos lygio, todėl neretai modeliuojant konkrečią pasirinktą sistemą sukuriamas prototipas „apgaubiantis“ panašiomis savybėmis ir tikslais pasižyminčias kitas sistemas. Šiame mokslo tiriamajame darbe nėra iškeliamas tikslas pilnai formaliai verifikuoti konkrečią adaptyvią kompiuterinę sistemą, bet atlikti tyrimą, kuriuo metu ištirti bendras tokių sistemų aibei savybes ir reikalavimus, apjungti skirtingas architektūrines strategijas bei įvertinti junginio pranašumus ir trūkumus. Sulig kiekviena iteracija kuriamas modelis bus tikslinamas siekiant tyrimo pabaigoje gauti detalų, bet tinkantį aibei adaptyvių kompiuterinių sistemų, modelį. Šį modelį galima būtų traktuoti kaip adaptyvių kompiuterinių sistemų klasės brėžinį, kurio pagrindu bus įmanoma suformuoti rekomendacijas bei įžvalgas.

## 3.1 Charakteristikos

Kritinių sistemų klasei dalinai priskiriamos adaptyvios kompiuterinės sistemos pasižymi:

- Distributyvumu (ang. *distributivity*) – darbo paskirstymu tarp kelių sistemos komponenčių;
- Klaidų ir gedimų toleravimu (ang. *fault tolerance*) – darbo tęsimu įvykus vienos (arba kelių) sistemos dalies gedimui (klaidos būsenai); siekiamą išlaikyti funkcionalumą proporcionaliai veikiančių komponenčių skaičiui – sistemos darbo prastėjimas yra pažingsninis (ang. *graceful degradation*);
- Prisitaikymu prie aplinkos pokyčių (ang. *adaptivness*) – aktualių aplinkos objektų pokyčių, veiksmų bei aplinkybių kaitos atpažinimu ir atitinkamu jų apdorojimu;

- Dinamine rekonfigūracija (ang. *dynamic reconfiguration*) – automatiniu sistemos konfigūracijos (parametrų, nustatymų, struktūros ir t.t.) atnaujinimu realiu laiku, reaguojant į pasikeitusią situaciją.

Bendru atveju, sugedus pavienei sudėtinei daliai, adaptyvi kompiuterinė sistema nenutraukia darbo – gedimas yra atpažįstamas bei atitinkamai apdorojamas. Siekiama maksimaliai sumažinti neigiamą įtaką visos sistemos darbo rezultatui, todėl darbas yra paskiriamas iš naujo, o sugedusios komponentės resursai yra paskirstomi tarp aktyvių dalių. Neveiksni dalis, priklausomai nuo jos tipo ir gedimo sudėtingumo, gali būti taisoma. Sistemos struktūrai gali priklausyti komponentė, kurios gedimas reikštų visos sistemos gedimą – tokiu atveju reikalingas išorinis (žmogaus, sistemą prižiūrinčio specialisto) įsitraukimas. Adaptyvi kompiuterinė sistema atlieka savo tiesioginį darbą bei reaguoja į pokyčius arba savo sudėtinių dalių gedimus tol, kol gedimai leidžia sistemai fiziškai veikti be išorinio įsikišimo.

## 3.2 Architektūra

Daugiaagentė (ang. *multi-agent*) sistema – sistema, kurią sudaro tarpusavyje sąveikaujančių autonominių agentų aibė. Agentai turi savo lokalų aplinkos supratimą (vaizdą, ang. *local view*), kuriuo remdamiesi priima sprendimus. Dažnu atveju lokalus vaizdas neprilygsta realiai situacijai – pvz. dėl vėluojančio arba neegzistuojančio (sugedus atitinkamai sistemos komponentei) vaizdo atnaujinimo. Jeigu aplinka yra padalinta į atskiras dalis, o kiekvieno agento veiklos sritis apima tik vieną arba kelias tos aplinkos dalis, pilno tikslaus vaizdo agentui nereikia – užtenka pakankamai tikslaus supratimo, kokia situacija esti jam paskirtoje konkrečioje dalyje. Bendru atveju agentai bendrauja asinchroniškai – išsiuntę pranešimą nepereina į laukimo būseną iki atsakymo sulaukimo, bet tęsia darbą.

Saviorganizuojanti daugiaagentė sistema (ang. *self-organising multi-agent system, MAS*) – decentralizuotų daugiaagenčių sistemų poaibis, kuriam priklauso sistemos, pasižyminčios mechanizmu leidžiančiu dirbti ir, esant būtinybei, pakeisti savo struktūrą (konfigūraciją) be konkretaus išorinio įsikišimo. Tokios sistemos yra pilnai autonominės ir geba prisitaikyti prie aplinkos kaitos. Viena iš tokios sistemos komponenčių gali būti vien adaptyvumą užtikrinanti posistemė, nedalyvaujanti tiesiogiai sistemos užduoties sprendime, bet palaikanti kitų komponenčių veikimą.

Kompiuterinės sistemos, kurias sudaro komponentės, turinčios įtakos kito tipo agentams ir turinčios tarpusavy komunikuoti (pvz. valdymo stotys, resursų paskirstytojai ir t.t.), susiduria su Bizantijos karvedžių problema (kitaip dar vadinama Bizantijos generolų problema, ang. *Bizantine generals problem*). Problema kyla, kai komunikuojančios komponentės (vadinamos karvedžiais) turi

pasiekti susitarimą (konsensusą), bet dėl vienos ar kelių sistemos dalių klaidos arba gedimo gali būti bendrai priimtas klaidingas sprendimas. Problema yra sudėtinga ir neturi universalaus sprendimo, tinkančio visiems sistemų tipams [DHS+03, LSP82].

### 3.2.1 Hierarchinė architektūra

Hierarchinės architektūros tipo sistemose kiekvienos komponentės pareigos yra aiškiai apibrėžtos. Nurodoma, kuri sistemos dalis priima sprendimus, priskiria užduotį kitai posistemei, o kuri – besąlygiškai atlieka gautus nurodymus ir praneša apie jų atlikimą.

Tyrime, kurį atliko I.Pereverzeva, E.Troubitsyna ir L.Laibinis [PTL12], buvo formaliai verifikuojama daugiarobotė sistema, kurios tikslas yra išvalyti jai priskirtą teritoriją. Sistema pasižymi nevienalyte (ang. *heterogenous*) architektūra – ją sudarė dviejų tipų robotai – stacionarios valdymo stotys bei judrūs robotai. Sistemoje yra griežta vertikali dviejų pakopų hierarchija. Valdymo stotys sudaro viršutinę pakopą, priskirdamos joms pavaldiems robotams darbą (esant būtinybei, perskirdamos darbą kitam robotui). Abiejų tipų robotai gali sugesti, todėl sistema turi būti pasiruošusi, įvykus gedimui, atlikti dinaminį rekonfigūravimą. Vienoje pakopoje esantys robotai yra tarpusavyje lygūs – todėl sistema neturi vieno prižiūrėtojo (ang. *supervisor*). Sistemos darbas yra koordinuojamas tarp valdymo stočių, joms tarpusavyje nuolat bendraujant ir besidalinant savo lokaliu situacijos įsivaizdavimu.

### 3.2.2 Saviorganizuojanti architektūra

Saviorganizuojančios architektūros sistemose jas sudarantys agentai (komponentės) veiksmų bei pareigų prasme yra tarpusavyje lygūs. Agentai negali nurodyti kitiems jų elgesio, o tik pasidalinti turima informacija. Remdamasis turimomis žiniomis bei nurodytomis taisyklėmis, agentas pats priima sprendimą, kuris tą akimirką atrodo teisingiausias. Mokslinėje literatūroje saviorganizuojanti architektūra dažnai yra vadinama „ieškančių maisto skruzdžių“ (ang. *foraging ants*) metodu. Skruzdžių kolonija bendradarbiauja siekdama rasti bei atnešti į lizdą maistą.

Turku, Sfakso bei Tulūzos universitetų mokslininkų atliktame tyrime [LTG+14] buvo formaliai modeliuojamas ir verifikuojamas skruzdžių kolonijos darbas. Vienas iš pagrindinių suformuluotų reikalavimų yra skruzdžių susidūrimo vengimas – t.y. užtikrinimas, kad esant pasirinkimui, skruzdė vengtų srities kur jau darbuojasi kita skruzdė. Straipsnio išvadose yra pabrėžiama, kad saviorganizuojančios daugiaagentės sistemos reikalauja įvairialypio tyrinėjimo, skirtingų metodų (pvz. stochastinės analizės) taikymo „siekiant nustatyti optimaliausius sistemos parametrus, kurie leistų sistemai pasiekti savo tikslus ne tik loginio teisingumo, bet ir našumo, patikimumo požiūriu“.

Tyrime, kuris buvo atliktas Linnaeus universitete [IfW12], buvo formaliai verifikuojama eismo greitkelyje stebėjimo sistema, susidedanti iš išmaniųjų kamerų. Kiekvienos kameros diapazonas yra ribotas, todėl jos yra tolygiai paskirstytos kelyje tam, kad būtų aprėptas maksimalus kelio plotas. Kameros turi aptikti eismo spūstis bei apie tai pranešti šviesoforų kontrolieriams ir vairuotojo pagalbos sistemoms. Šios užduoties sprendime decentralizacija yra suprantama – valdymo centras šioje situacijoje atliktų tik „papildomos stotelės“ arba kliūties (ang. *bottleneck*) vaidmenį, prailgindamas laiką tarp spūsties pastebėjimo ir kameros pranešimo gavimo. Tačiau esant didelei eismo spūsčiai (pastebimai daugiau nei vienai kamerai), ją filmuojančios kameros jungiasi į vieną bendrą organizaciją, kuri sudaro detalų spūsties apibrėžimą. Transporto priemonėms pajudėjus (nustojus egzistuoti spūsčiai) kameros atsiskiria nuo organizacijos ir tęsia darbą atskirai. Tokiu būdu dinamiškai yra kuriama trumpalaikė minimali hierarchija tarp kelių sistemos komponenčių.

## 3.3 Reikalavimai

Pagrindinis ir intuityviai suprantamas reikalavimas, keliamas bet kuriai adaptyviai kompiuterinei sistemai – tikslo pasiekimas nepriklausomai nuo aplinkos kaitos ar gedimų skaičiaus. Sistema turi gebėti tiksliai įvertinti situacijos kaitą bei atitinkamai į ją sureaguoti. Modeliuojant šį reikalavimą galima apibrėžti kitaip – kiekvienas sistemos veikimo scenarijus (vykdymas) galiausiai pasiekia būseną, žyminčią pilną užduoties atlikimą.

Tarp bendrų visoms adaptyvioms kompiuterinėms sistemoms reikalavimų yra ir iš pirmojo bendro reikalavimo iš dalies išplaukiantis klaidų toleravimas – bet kuriuo metu viena ar kelios sistemos dalys gali sugesti (nutraukti darbą, nustoti veikti) apibrėžtam laiko tarpui arba visam laikui, tačiau tai neturi reikšti visos sistemos darbo nutraukimą. Sugedus komponentei sistema privalo atlikti dinaminį rekonfigūravimą, perorganizuoti savo struktūrą (pvz. perdalinti darbą tarp veikiančių komponenčių) siekiant išlaikyti sklandų darbą bei galiausiai pilną atlikimą.

Vienu iš konkrečiai tiriamai adaptyvių kompiuterinių sistemų poaibiui keliamų reikalavimų yra komponenčių autonomiškumas. Sistemoje neturi egzistuoti bendra, visą sistemos darbą kontroliuojanti komponentė. Sistema privalo sudaryti kelios posistemės, tarp kurių yra paskirstytas darbas bei resursai, kuriuos kontroliuos ir koordinuos tik ta viena sistemos dalis. Komponentės privalo komunikuoti tarpusavyje siekiant užtikrinti anksčiau įvardintų reikalavimų tenkinimą (pvz. klaidų toleravimą – bendraudamos tarpusavyje posistemės atliks sistemos rekonfigūraciją bei bendrai reaguos į kurios nors dalies gedimą)

## 3.4 Taikomi metodai

Apžvelgtuose tyrimuose buvo taikomi skirtingi formalaus verifikavimo metodai bei įrankiai. Teritorijos valymo sistema ir ieškančių maisto skruzdžių lizdas buvo verifikuojami taikant teoremų įrodymą Rodin aplinkoje (Event-B). Teritorijos valymo sistemos tyrimo pratęsime [TPT+13] buvo taikomas PRISM modelių patikrintojas siekiant tikimybiškai įvertinti pasirinkto tikslo pasiekiamumą. Eismo greitkelyje stebėjimo sistema buvo modeliuojama naudojant standartinį UPPAAL modelių patikrinimo paketą. Bolonijos universitete atliktame tyrime [CaV09] buvo prieita išvados, kad stochastinis simuliavimas ir tikimybinis modelių patikrinimas yra itin naudingi tiriant saviorganizuojančių sistemų emergencinėms (ang. *emergent* – visos sistemos, bet ne jos atskirų komponenčių) savybėms [BDT99].

Ankstesniame darbe atliktame tyrime [Dau21] adaptyvi kompiuterinė sistema buvo formaliai verifikuojama taikant modelių patikrinimo metodą, modelį aprašant formalios specifikacijos kalba $TLA^+$. Buvo įrodyta, kad sistema, pasižyminti kokybinėmis savybėmis veikia korektiškai (pvz. nepriklausomai nuo komponenčių gedimo dažnio visada pilnai atlieka darbą). Šiame darbe bus siekiama tyrimą praplėsti. Planuojama nuodugniau analizuoti sistemos laiko savybes, todėl bus naudojamas kitas formalaus verifikavimo įrankis – UPPAAL SMC. Neegzistuojant tiesioginiam vertėjui iš $TLA^+$ į UPPAAL modelį reikės sukurti iš naujo, nuo pagrindų, remiantis ankstesniame tyrime sukurtu modeliu kaip teoriniu žinių pagrindu. Tačiau uždavinys sukurti ir sėkmingai formaliai verifikuoti adaptyvios kompiuterinės sistemos modelį nėra keliamas – šis tikslas jau buvo realizuotas anksčiau ir yra suprantamas kaip savaime suprantamas reikalavimas naujam tyrimo etapui. Šiame darbe bus tyrinėjamos laiko aspektas – kaip greitai sistema reaguoja į pokyčius (gedimus) bei atlieka dinaminę rekonfigūraciją ir t.t. Lygiagrečiai panaši sistema bus simuliuojama AnyLogic įrankio pagalba. Tai nėra esminė tyrimo dalis, tačiau papildys tyrimą, leis analizuoti problemas iš kitos perspektyvos.

## 3.5 Išvados

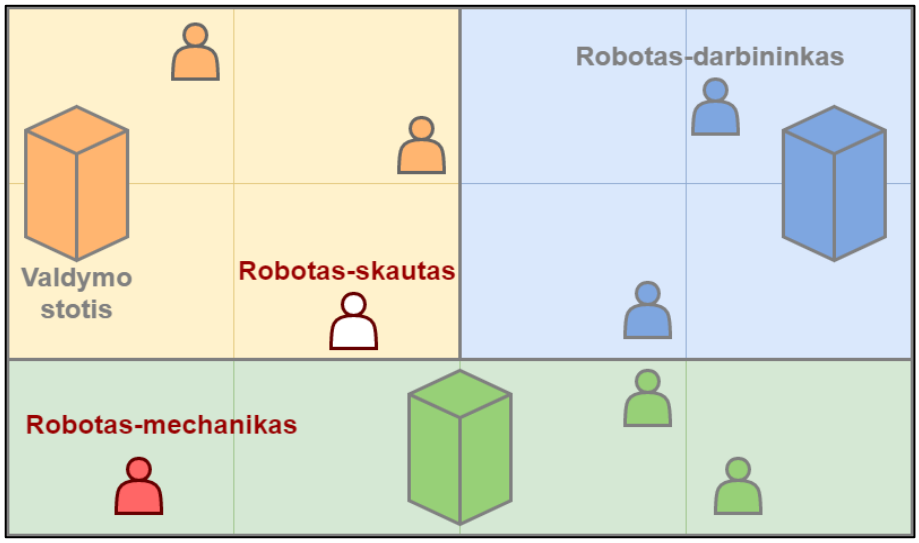
Remiantis atlikta mokslinių tyrimų ir straipsnių, parašytų panašios specifikos kompiuterinių sistemų formalaus verifikavimo tema, apžvalga, buvo apibrėžti keliami adaptyviai kompiuterinei sistemai reikalavimai, jos charakteristikos ir bendri bruožai. Literatūros analizės metu taip pat buvo identifikuoti naudojami panašių sistemų verifikavimui metodai bei prieita išvados, jog siekiant tyrinėti sistemos laiko savybes tinkamiausias metodas yra simuliavimu besiremiantis statistinis modelių patikrinimas, kuris leis interaktyviai tyrinėti modelio kaitą bei gauti pakankamai duomenų tolesnei

analizei ir samprotavimams pagrįsti. Besiremiant ankstesniame skyriuje pristatyta formalaus verifikavimo įrankių apžvalga adaptyvios kompiuterinės sistemos formaliam modeliavimui ir verifikavimui pasirinktas UPPAAL bei statistinį modelių patikrinimą implementuojantis plėtinys UPPAAL SMC.

Baigiamajame magistro darbe apjungiami du sistemos organizavimo modelius – hierarchinį (besiremiantį aiškiu pareigų bei įtakos darbui pasiskirstymu) bei saviorganizuojantį (pvz. besimaitinančių skruzdžių – kuriame, iš esmės, visi agentai yra lygūs). Sistemą sudaro skirtingų tipų komponentės, sudarančios kelis sistemos sluoksnius – tačiau tame pačiame sluoksnyje esančių agentų autonomiškumo lygis bus identiškas. Tyrimo metu analizuojami tokio architektūrinio hibrido pranašumai bei trūkumai – atkreipiant dėmesį į tai, ar savybes yra „paveldimos“ iš kurios nors egzistuojančios architektūros sprendimo, ar atsiranda dėl apjungimo. Tikimybinis sistemos modelio vertinimas yra ypatingai pravartus, kadangi leidžia tiksliai įvertinti naujos architektūros charakteristikas.

# 4 Teritoriją valanti robotų sistema

Šio darbo tyrimo metu buvo formaliai modeliuojama ir verifikuojama teritoriją valanti robotų sistema (žr. *Pav. 8*). Sistema privalo pilnai sutvarkyti jai priskirtą teritoriją. Sistema priklauso adaptyvių kompiuterinių sistemų poaibiui, todėl jai keliami šiame darbe pristatyti bendri reikalavimai – pilnas darbo atlikimas nepriklausomai nuo situacijos kaitos, klaidų toleravimas, autonomiškumas.




*Pav. 8. Adaptyvios kompiuterinės sistemos schema*

Teritoriją valančios robotų sistemos idėja remiasi anksčiau atliktais tyrimais, kuriose panašios specifikos sistema buvo formaliai verifikuojama tiriant tikimybines charakteristikas PRISM modelių patikrintoju [TPT+13] bei funkcines savybes Event-B [PTL12] ir TLA$^+$ [Dau21] įrankiais. Šiame tyrime analizuojamą sistemą galima laikyti nauja teritoriją valančių robotų sistemos versija, kadangi ji yra papildyta naujomis komponentėmis.

## 4.1 Architektūra

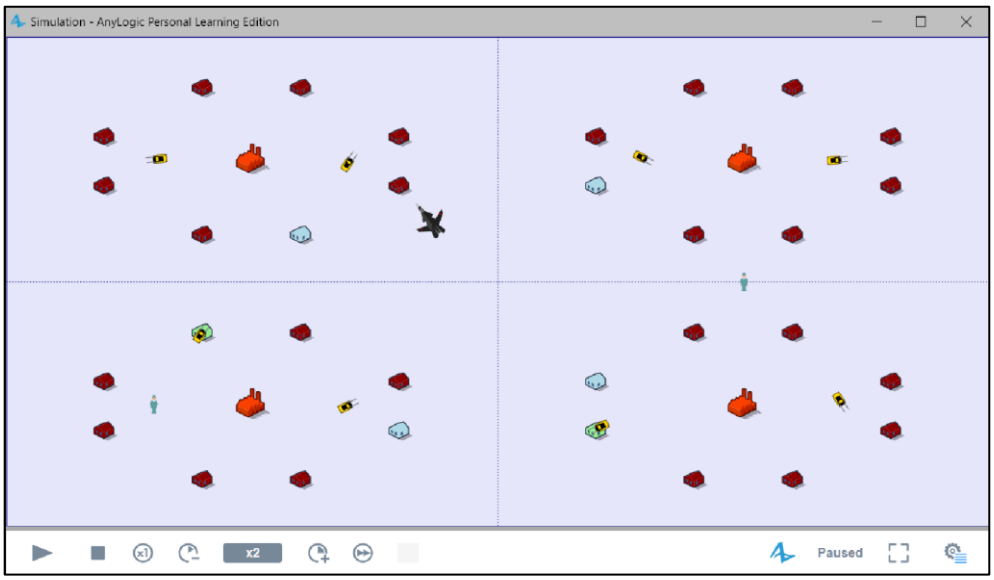
Tiriamos adaptyvios kompiuterinės sistemos architektūrą sudaro dviejų organizavimo modelių – hierarchinio ir saviorganizuojančio – hibridas. Sistemoje egzistuoja hierarchija, tačiau ji tik dalinai griežta; dalis komponenčių hierarchijai nepriklausys visiškai ir yra sistemos kontekste autonomiški (sprendimus priima patys, savo nuožiūra).

Adaptyvią kompiuterinę sistemą (žr. *Pav. 9*) sudaro keturių tipų robotai:

- Valdymo stotis – aukščiausio hierarchijos sluoksnio statinė komponentė. Jai yra pavaldūs robotai-darbininkai (kiekviena valdymo stotis turi aibę robotų, priklausančių tik jai). Kiekviena stotis turi savo lokalų teritorijos ir situacijos įsivaizdavimą, kuriuo

remdamasi priskiria darbus robotams-darbininkams;

- Robotas-darbininkas – judrus mechaninis aparatas. Kiekvienas robotas-darbininkas yra priskirtas vienai konkrečiai valdymo stočiai, iš kurios gauna darbą (sektoriaus kurį reikia išvalyti identifikacinį numerį). Gavęs užduotį, robotas-darbininkas vyksta į nurodytą sektorių ir ten atlieka valymą. Pilnai atlikęs darbą robotas-darbininkas siunčia pranešimą valdymo stočiai, su kuria yra susietas;
- Robotas-mechanikas – taisantis sugedusius robotus-darbininkus bei valdymo stotis mechanizmas. Sugedus robotui-darbininkui arba valdymo stočiai ir tai pastebėjus skautui, robotas-mechanikas gauna pranešimą. Gavęs pranešimą, robotas-mechanikas vyksta į nurodytą lokaciją bei taiso sugedusią komponentę;
- Robotas-skautas – žvalgo rolę atliekantis aparatas. Robotas tyrinėja teritoriją, atsitiktinai pasirinkdamas tikslą – komponentę, kurios būseną tikrins. Pastebėjęs roboto-darbininko arba valdymo stoties gedimą, robotas-skautas apie tai praneša robotui-medikui.




*Pav. 9. Adaptyvios kompiuterinės sistemos modelis AnyLogic aplinkoje*

## 4.2 Reikalavimai

Dėl komercinių priežasčių gauti konkrečios kompiuterinės sistemos reikalavimų dokumento pavyzdį dažnu atveju nėra įmanoma, o viešai pasiekiami aprašymai yra pernelyg abstraktūs ir bendri. Remiantis pristatytais ir analizuotais moksliniais tyrimais [CaV09, IfW12, LTG+14, PTL12], kurių metu buvo tiriamos ir modeliuojamos panašios arba artimos analizuojamai adaptyvios sistemos, suformuluoti pagrindiniai sistemos reikalavimai:

**(R1)** Adaptyvios kompiuterinės sistemos darbo tikslas – pilnai išvalyti visą jai priskirtą teritoriją;

**(R2)** Teritorija, kurią apima adaptyvios kompiuterinės sistemos veikla, yra išskaidyta į nesikertančius sektorius, kurie gali būti apjungiami į zonas.

**(R3)** Adaptyvią kompiuterinę sistemą sudaro keturių tipų, skirtingų hierarchijos lygių, robotai:

- Valdymo stotis;
- Robotas-darbininkas;
- Robotas-skautas;
- Robotas-mechanikas.

**(R4)** Kiekvienai valdymo stočiai yra priskirti:

- Robotų-darbininkų poaibis;
- Sektorių poaibis (zona).

**(R5)** Robotai-mechanikai ir robotai-skautai nėra tiesiogiai susieti su konkrečia valdymo stotimi;

**(R6)** Kiekviena valdymo stotis turi savo lokalų vaizdą – įsivaizdavimą apie tai, kokia yra kiekvieno teritorijos sektoriaus būsena;

**(R7)** Valdymo stotis priskiria darbą robotui-darbininkui pateikdama pasirinkto neišvalyto (pagal turimą lokalų vaizdą) sektoriaus identifikacinį numerį;

**(R8)** Valdymo stotis atnaujina savo lokalų vaizdą pagal gautą iš roboto-mechaniko pranešimą;

**(R9)** Valdymo stotys periodiškai tarpusavyje bendrauja, atnaujindamos savo lokalų vaizdą pagal gautus pranešimus;

**(R10)** Valdymo stotis gali sugesti bet kurio metu ir būti sutaisoma neribotą kiekį kartų;

**(R11)** Valdymo stočiai būnant gedimo būsenoje, kita tuo metu veikianti valdymo stotis, pagal kurios lokalų vaizdą nėra išvalyti visi sektoriai, gali perimti sugedusios valdymo stoties robotus-darbininkus ir sektorius;

**(R12)** Jeigu sutaisyta valdymo stotis neturi su ja susietų robotų-darbininkų ir sektorių, ji pereina į stebėjimo (ang. *monitoring*) režimą, kurio metu stebi kitas valdymo stotis bei periodiškai siunčia užklausas dėl robotų-darbininkų ir sektorių perskyrimo;

**(R13)** Tuo pačiu metu robotui-darbininkui gali būti priskirtas tik vienas sektorius;

**(R14)** Atlikęs priskirto sektoriaus valymą, robotas-darbininkas siunčia pranešimą apie pilnai išvalytą sektorių valdymo stočiai, kuriai, pagal tuometinį roboto-darbininko lokalų vaizdą, jis yra priskirtas;

**(R15)** Robotas-darbininkas gali sugesti ir būti sutaisomas neribotą kiekį kartų;

**(R16)** Robotas-darbininkas turi ribotą energijos, kurį yra naudojama pasiekti užduoties tikslą (sektorių) bei atlikti darbą (išvalyti), kiekį;

**(R17)** Gavęs užduotį (priskirto sektoriaus numerį), robotas-darbininkas apskaičiuoja energijos, reikalingos atlikti užduočiai, kiekį;

**(R18)** Robotas-darbininkas kiekį energijos, reikalingos atlikti užduočiai, apskaičiuoja sudėdamas energijos, reikalingos pasiekti priskirtą sektorių iš dabartinės roboto pozicijos, išvalyti maksimaliai užterštą sektorių bei iš to sektoriaus pasiekti su roboto-darbininku susietą valdymo stotį, kiekį;

**(R18)** Apskaičiavęs reikiamos užduočiai atlikti energijos kiekį, robotas-darbininkas palygina skaičių su turimu energijos kiekiu:

**(R18.1)** Jeigu robotas-darbininkas turi daugiau energijos nei reikiama, jis atlieka darbą (pradeda vykti sektoriaus link);

**(R18.2)** Jeigu robotas-darbininkas neturi pakankamai energijos pilnai atlikti darbą ir grįžti į valdymo stotį, su kuria yra susietas, robotas atsisako darbo ir apie tai informuoja valdymo stotį;

**(R18.3)** Atsisakęs darbo, dėl per mažo turimos energijos kiekio, robotas-darbininkas keliauja prie su jo susietos valdymo stoties. Energija yra įkraunama iki maksimalios reikšmės.

**(R19)** Aptikęs sugedusį robotą-darbininką arba valdymo stotį, robotas-skautas siunčia pranešimą robotui-mechanikui;

**(R20)** Robotas-mechanikas taiso sugedusias komponentes (valdymo stotis ir robotus-darbininkus);

**(R21)** Tuo pačiu metu robotui-mechanikui gali būti priskirtas tik vienas sugedęs robotas-darbininkas arba valdymo stotis;

**(R22)** Komunikacija tarp sistemos komponenčių (robotų) yra vykdoma siunčiant pranešimus;

**(R23)** Zona yra laikoma išvalyta, kai kiekvienas su ta zona susietas sektorius yra išvalytas;

**(R24)** Teritorija yra laikoma išvalyta, kai kiekviena teritorijos zona (kiekvienas sektorius) yra išvalyta;

**(R25)** Valdymo stotis užbaigia darbą tada, kai pagal nuosavą lokalų vaizdą visa teritorija yra išvalyta;

**(R26)** Paskutinei valdymo stočiai užbaigiant darbą, kito tipo robotams (darbininkams, medikams, skautams) siunčiamas pranešimas užbaigti darbą;

**(R27)** Adaptyvi kompiuterinė sistema baigia darbą, kai visa teritorija yra išvalyta.

# 5 Adaptyvios kompiuterinės sistemos verifikavimas

Baigiamojo magistro darbo metu buvo atliktas adaptyvios kompiuterinės sistemos formalaus verifikavimo tyrimas. Tyrimas susideda iš sistemos formalaus modeliavimo ir verifikavimo taikant modelių patikrinimo metodus. Sukurtas abstraktus sistemos modelis formaliai verifikuotas taikant klasikinį modelių patikrinimo metodą. Įrodžius esminių savybių ir reikalavimų patenkinimą modelis tobulinamas bei verifikuojamas taikant statistinį modelių patikrinimą; tiriamos kompiuterinės sistemos laiko savybės. Tyrimo pabaigoje trumpai yra nagrinėjamas simuliavimo panaudojimas gilesnei verifikuojamo objekto analizei.

## 5.1 Formalus modeliavimas

Tyrime verifikuojamą adaptyvią kompiuterinę sistemą sudaro keturių tipų robotai:

- valdymo stotys;
- robotai-darbininkai;
- robotai-mechanikai;
- robotai-skautai;

du vidiniai sistemos mechanizmai, kurie gali būti traktuojami kaip programinė įranga arba operacinės sistemos dalis:

- resursų perskirstymas;
- sistemos komponenčių išjungimas;

bei dviejų robotų tipų – valdymo stočių ir robotų-darbininkų – gedimo procesas.

Dėl modeliavimo kalbos specifikos visos duomenų struktūros yra deklaruojamos globaliai. Globalių modelio kintamųjų deklaracijos fragmentas yra pateikiamas žemiau (žr. *Pav. 10*); pilna deklaracija kartu su modeliais yra pateikta prieduose (*Nr. 1 ir 2*). Kai kurių robotų (valdymo stočių, darbininkų ir mechanikų) šablonus papildo struktūros (*struct*), susidedančios iš įvairaus tipo laukų, pvz. *boolean* tipo lauko *active*, žyminčio, ar šiuo metu robotas yra veiksnus (*false* reikšmės atveju laikoma, jog robotas yra sugedęs), arba sektoriaus tipo *posX* ir *posY*, nurodančių, kurioje šiuo metu pozicijoje yra robotas. Procesų prieiga prie duomenų yra ribojama – pvz. valdymo stotis tiesiogiai gali pasiekti tik savo lokalų vaizdą (masyvas *localView*, pagal indeksą atitinkantį sektoriaus identifikacinį numerį – R6). Kitos valdymo stoties lokalų vaizdą komponentė gali matyti tik atitinkamais atvejais.

Dalis duomenų struktūrų nurodo ryšius tarp skirtingo tipo robotų, teritorijos. Sektoriaus priklausymą valdymo stočiai apibrėžia *sectorStation* – masyvo elementas žymi konkretų sektorių (indeksas atitinka identifikacinį numerį), o elemento reikšmė – valdymo stotį, kuriai sektorius yra

priskirtas. Masyvas *workerSector* žymi, koks sektorius yra šiuo metu yra priskirtas pasirinktam robotui-darbininkui.

Sektorių užterštumas yra modeliuojamas skaitine verte masyve *sectorStatus*. Kiekvienas masyvo elementas žymi konkretų sektorių, reikšmė nurodo kiek laiko vienetų užimtų valymo darbai. Elementui pasiekus nulinę reikšmę, sektorius yra laikomas išvalytu. Masyve *sectorAssigned* yra saugoma informacija, ar pasirinktas sektorius jau yra arba buvo priskirtas robotui-darbininkui.

```
const int NUM_OF_STATIONS =  4;
const int NUM_OF_WORKERS  = 12;
const int TERRITORY_LEN   =  5;

const double normRate  = 1.0;
const double errorRate = 0.5;
const double saveRate  = 0.25;

broadcast chan work[worker_t],
               done[worker_t];

const int WORKER_ENERGY = 25;

typedef struct
{
    sector_t  posX;
    sector_t  posY;
    station_t station;
    bool      assigned;
    bool      started;
    bool      active;
    int       energy;
    bool      refill;
    bool      terminated;
} robotWorker_t;

robotWorker_t worker[NUM_OF_WORKERS] = { {0, 1, 0, 0, 0, 1, WORKER_ENERGY, 0, 0},
```

*Pav. 10. Kompiuterinės sistemos modelio globalių kintamųjų deklaracijos fragmentas*

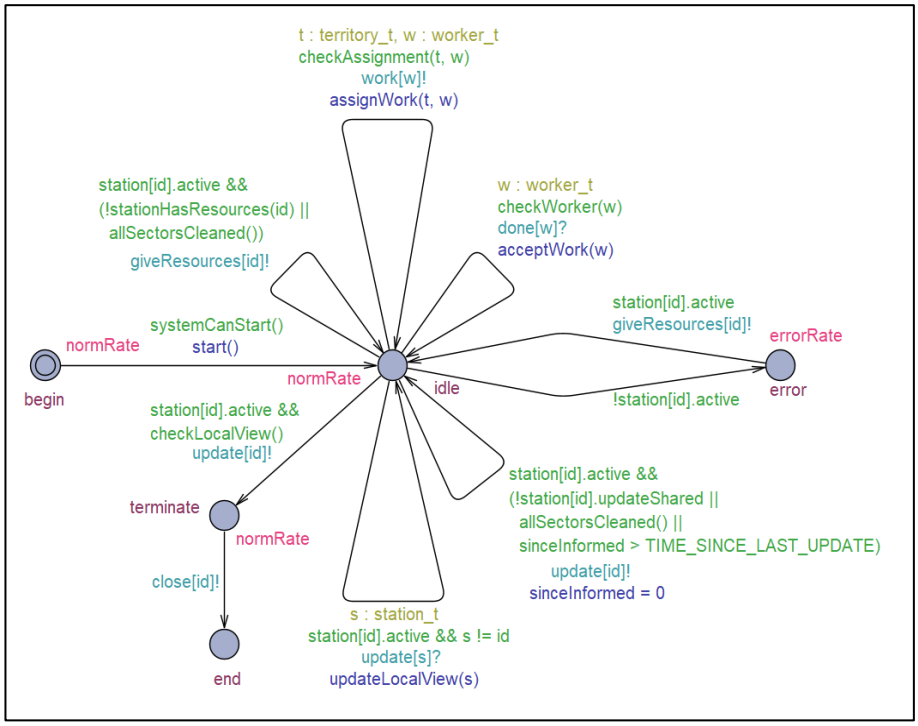
### 5.1.1 ControlStation

Valdymo stotį atitinkantis procesas yra modeliuojamas šablonu *ControlStation* (žr. *Pav. 11*). Iš pradinės padėties *begin*, pradėjus veikti sistemos reagavimą į gedimus užtikrinančioms komponentėms, su eksponentiniu koeficientu *normRate* yra pereinama prie padėties *idle*. Tai yra centrinė proceso būsena, su kuria yra susijusios esminės proceso atliekamos operacijos:

- Valdymo stočiai pasirinkus tinkamus robotą-darbininką bei sektorių (tenkinančius *checkAssignment* metode aprašytas sąlygas, kaip pvz. „robotui nėra priskirtas joks sektorius“ ir „sektorius nepriskirtas jokiam robotui“, „pagal lokalų valdymo stoties vaizdą sektorius nėra išvalytas“ ir t.t.) konkrečiam robotui-darbininkui pagal jo identifikacinį

numerį yra siunčiamas sinchronizacijos pranešimas (*work[w]!*);

- Valdymo stočiai būnant nesugedusiai bei gavus iš jai pavaldaus roboto-darbininko pagal jo identifikacinį numerį pranešimą apie sėkmingą darbo atlikimą (*done[w]?*), modifikuojama valdymo stoties lokali informacija (atnaujinamas lokalus vaizdas, pasižymima, kad robotas nuo to laiko tarpo nebeturi priskirto darbo, o kitos valdymo stotys dar nėra informuotos apie naujausią sektoriaus išvalymą);
- Valdymo stočiai būnant aktyviai bei žinant, kad informacija apie naujausią sektoriaus išvalymą su kitomis stotimis nebuvo pasidalinta, arba visiems su valdymo stotimi susietiems sektoriams būnant pilnai išvalytiems, siunčiamas pranešimas pagal valdymo stoties identifikacinį numerį (*update[id]!*; R9). Pradinės sistemos versijos verifikavimo metu, pastebėjus būtinybę užtikrinti ciklinį atnaujinimo siuntimą, modelis buvo praplėstas laikmačiu *sinceInformed*, skaičiuojančiu laiką nuo paskutinio lokalaus vaizdo pasidalinimo. Briaunos saugas buvo papildytas sąlyga: *turi praeiti daugiau laiko nuo paskutinio lokalaus vaizdo pasidalinimo nei nurodyta parametre TIME_SINCE_LAST_UPDATE*;
- Valdymo stočiai būnant aktyviai bei sulaukus pranešimo iš kitos valdymo stoties apie naujausią sektoriaus išvalymą (*update[id]?*), atnaujinamas lokalus vaizdas (metodas *updateLocalView*);
- Valdymo stočiai sugedus (nebūnant aktyviai) pereinama į klaidos padėtį *error*; iš jos grįžtama į padėtį *idle* su koeficientu *errorRate* kai valdymo stotis vėl tampa aktyvi (kai ją sutaiso robotas-mechanikas) bei siunčiama užklausa resursų valdymo programai dėl robotų ir sektorių priskyrimo pagal valdymo stoties identifikacinį numerį (*giveResources[id]!*);
- Valdymo stočiai būnant aktyviai ir neturint robotų-darbininkų ir/arba neišvalytų sektorių, siunčiama užklausa resursų valdymo programai dėl robotų ir sektorių priskyrimo (*giveResources[id]!*);
- Valdymo stočiai būnant aktyviai ir pagal lokalų vaizdą įsivaizduojant, jog visa sistemai priskirta teritorija yra pilnai išvalyta, pereinama į padėtį *terminate*; pakeliui siunčiamas paskutinis lokalaus vaizdo atnaujinimo pranešimas kitoms valdymo stotims. Pasiekus padėtį *terminate* yra siunčiamas pranešimas apie darbo pabaigą sistemos komponenčių išjungimo mechanizmui (*close[id]!*) ir pereinama į būseną *end* – valdymo stotis baigia darbą.



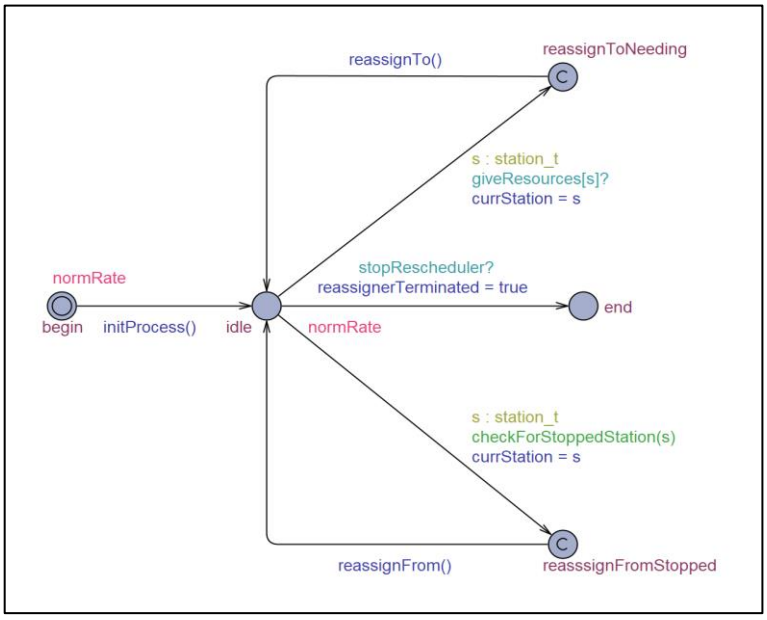

*Pav. 11. Valdymo stotį modeliuojantis šablonas*

#### 5.1.1.1 ReassignmentMechanism

Skirtingai nuo *ControlStation*, žyminčio konkrečią fizinę sistemos komponentę, *ReassignmentMechanism* šablonas modeliuoja valdymo stočių resursų (robotų-darbininkų ir sektorių) valdymo programą (žr. *Pav. 12*). Procesą galima interpretuoti kaip valdymo stočių resursų pasidalinimo funkcionalumo abstrakciją; vietoje to, kad valdymo stotys pačios įvertintų, kuri galėtų perduoti kiek robotų-darbininkų arba sektorių, ši užduotis yra perduodama programai operacinėje sistemoje. Antra vertus, toks sprendimas sukuria inkapsuliaciją, kadangi valdymo stotys neturi tiesioginės prieigos prie globalių (kompiuterinės sistemos kontekste) kintamųjų bei padeda išspręsti Bizantijos karvedžių problemą. Iš pradinės padėties *begin* pereinama į padėtį *idle*, iš kurios:

- Gavus užklausą iš valdymo stoties (*giveResources[s]?*), pereinama prie įsipareigojusios padėties *reassignToNeeding*, iš kurios grįžtant į laukimo padėtį *idle* atliekamas *reassignTo* metodas – t.y. iš pradžių ieškoma sugedusių valdymo stočių; jų neradus, iš aktyvių proporcionaliai valdymo stočių skaičiui yra perimami arčiausiai užklausą atsiuntusios stoties esantys robotai-darbininkai ir sektoriai;
- Pastebėjus stoties neaktyvumą (gedimo arba darbo užbaigimo atveju), pereinama į įsipareigojusią padėtį *reassignFromStopped*, iš kurios grįžtant į laukimo padėtį *idle* perskiriami resursai kitai aktyviai valdymo stočiai (*reassignFrom*);

- Gavus pranešimą iš išjungimo mechanizmo, pasiekiama padėtis *end* – programa baigia darbą.



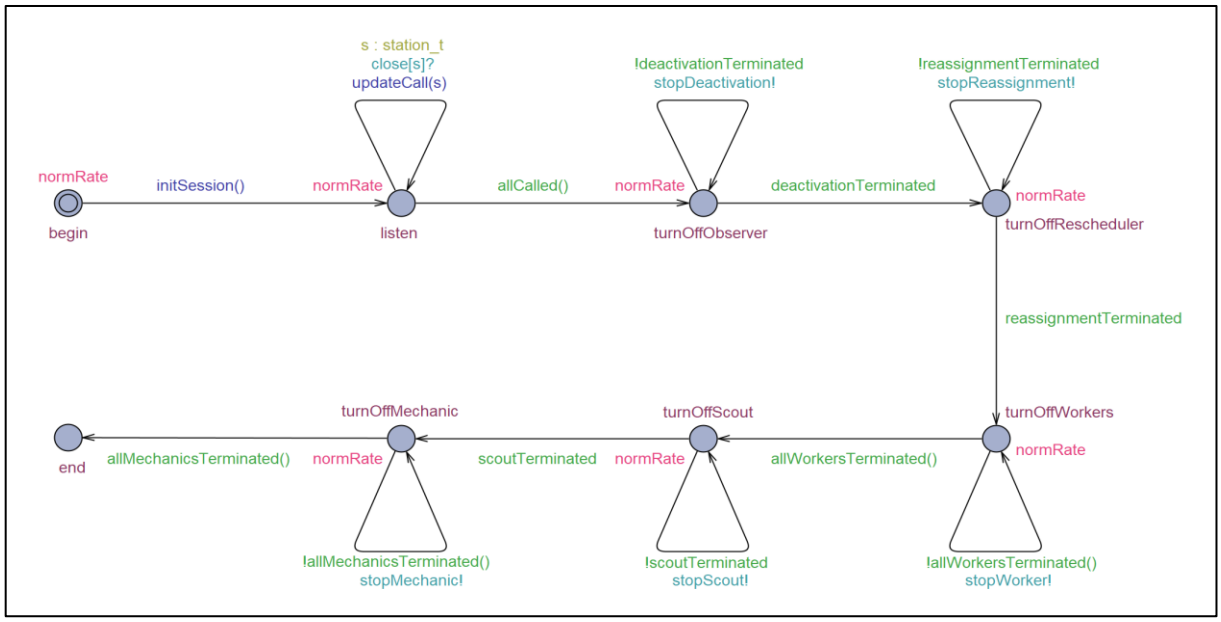

*Pav. 12. Resursų valdymo programą modeliuojantis šablonas*

#### 5.1.1.2 TerminationMechanism

Panašiai kaip ir *RescheduleMechanism*, *TerminationMechanism* šablonas (žr. *Pav. 13*) modeliuoja vieną iš valdymo stočių procesų – šiuo atveju tai yra užtikrinimas, jog visos kompiuterinės sistemos komponentės, pilnai išvaliusios visą teritoriją, užbaigs darbą. Su eksponentiniu koeficientu *normRate* pradinė padėtis *begin* yra paliekama ir pereinama į padėtį *listen*. Gavus pranešimą iš pirmos valdymo stoties apie darbo užbaigimą, padėtis yra paliekama bei atnaujinamas lokalus masyvas, kuriame saugoma informacija, ar pasirinkta valdymo stotis baigė darbą, ar dar ne; jeigu dar ne visos valdymo stotys baigė darbą, kilpa grįžtama į tą pačią padėtį; jei pranešimą atsiuntė paskutinė valdymo stotis (tenkinamas patikrinimas *allStationsTerminated*) pereinama į padėtį *turnOffDeactivation*. Šioje būsenoje yra liekama tol, kol „gedimo mechanizmas“ nėra sustabdytas – tada pereinama į padėtį *turnOffRescheduler*, kur yra stabdoma resursų valdymo programa. Sustabdžius programą (kintamajam *managerTerminated* įgavus reikšmę true) pereinama prie robotų-darbininkų išjungimo padėties *turnOffWorkers.*

Kol visi robotai-darbininkai nebaigia darbo yra siunčiamas pranešimas *stopWorker!* – verta pabrėžti, jog skirtingai nuo daugumos sinchronizavimo pranešimų modelyje, šis pranešimas nėra suasmenintas – o kadangi dėl statistinio modelių patikrinimo specifikos visi sinchronizavimo kanalai privalo būti transliacijos (*broadcast*) tipo – vienu metu tą patį pranešimą gali gauti keli robotai-

darbininkai. Įsitikinus, jog visi robotai-darbininkai baigė darbą (*allWorkersTerminated*), pereinama į būseną, stabdančią žvalgybos funcionalumą (*turnOffScout*), iš kurios pereinama į roboto-mechaniko stabdymo būseną *turnOffMechanic*. Galiausiai, robotui-mechanikui užbaigus darbą, darbą užbaigia ir pats išjungimo mechanizmas.




*Pav. 13. Sistemos išjungimo mechanizmą modeliuojantis šablonas*

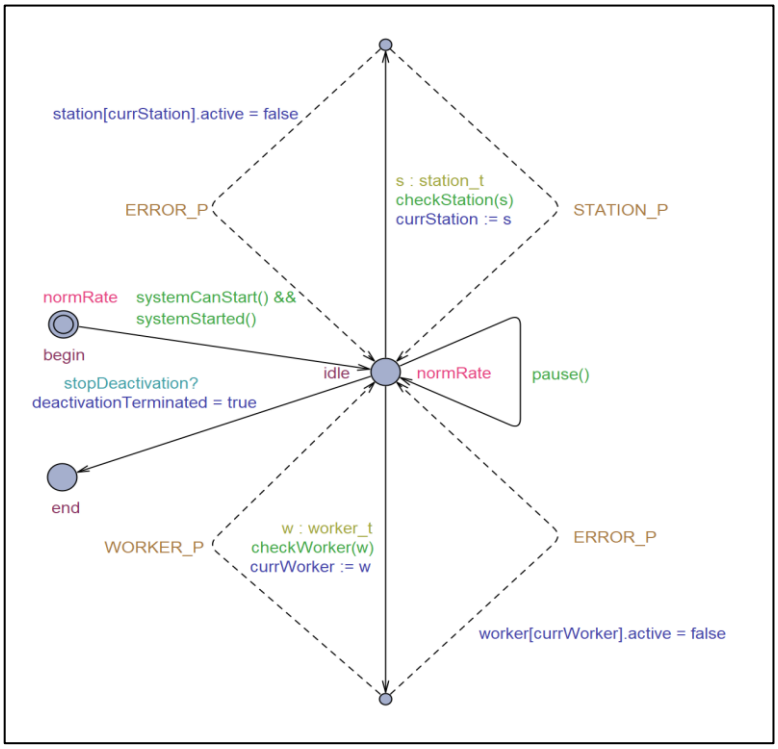
### 5.1.2 DeactivationProcess

*DeactivationProcess,* kaip *RescheduleMechanism* ir *TerminationMechanism,* modeliuoja ne konkretų robotą (sistemos komponentę), bet procesą (žr. *Pav. 14*). Tačiau, skirtingai nuo valdymo stoties mechanizmų, *DeactivationProcess* atitinka realaus pasaulio procesą ir tiesiogiai veikia valdymo stotis ir robotus-darbininkus – modeliuodamas jų gedimus. Pradedant *begin* padėtyje, su eksponentiniu koeficientu *normRate* pereinama į padėtį *idle*. Iš laukimo būsenos, su tuo pačių eksponentiniu koeficientu, galima:

- Pasirinkus veikiančią valdymo stotį pereiti į šakos tašką (ang. *branch point*) – iš kurio, remiantis priskirtais šakoms svoriams, pasirinkama viena iš briaunų. Valdymo stoties atveju nurodomas parametrų *STATION_P* ir *ERROR_P* santykis – tačiau tai turėtų būti labiau traktuojama kaip tikimybė kad tuo konkrečiu momentu valdymo stoties aktyvumas bus pakeistas (bus modeliuojamas gedimas; pasirinktą valdymo stotį atitinkančiam kintamajam *stationActive* priskiriama reikšmė *false*). Abiejų briaunų pasirinko atveju grįžtama į padėtį *idle* (pasirinkus kitą briauną jokie papildomi veiksmai nėra atliekami);
- Pasirinkus veikiantį robotą-darbininką, atliekamas analogiškas valdymo stoties pasirinko

atveju scenarijus – šiuo atveju santykis yra *ERROR_P:WORKER_P*;

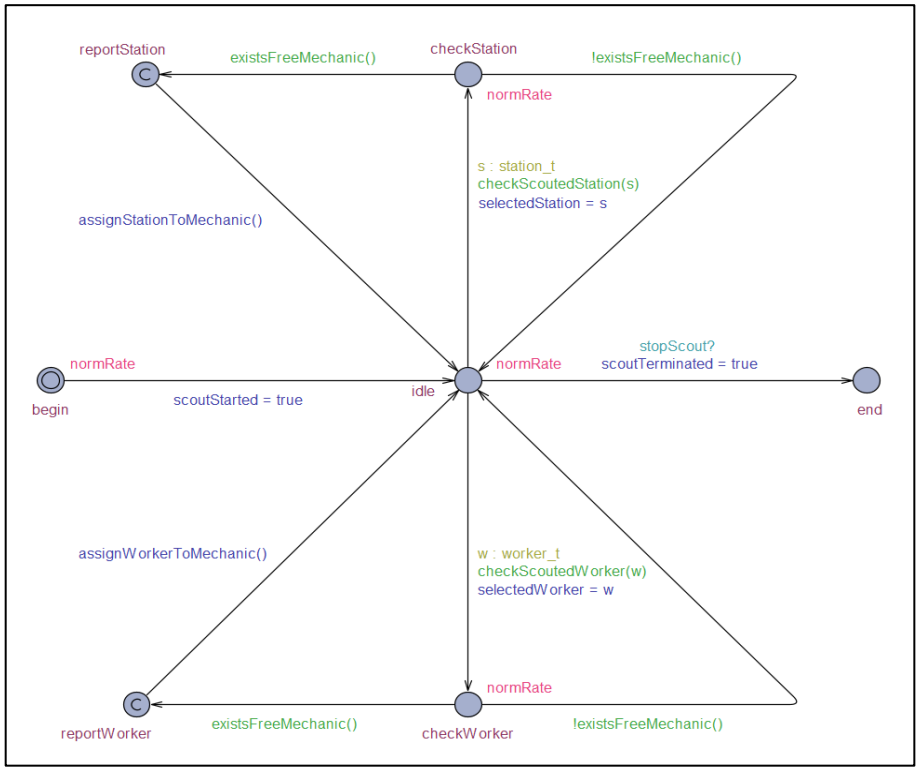
- Sugedus tiek visoms valdymo stotims, tiek robotams-darbininkams (tenkinama sąlyga, apibrėžta *pause*), pereinama kilpa ir grįžtama į padėtį *idle*;
- Gavus pranešimą iš išjungimo mechanizmo, pereinama į padėtį *end* – daroma prielaida, kad visoms valdymo stotims baigus darbą, likę robotai naudoja minimalų energijos kiekį bei nebegenda – gedimo pilnai atlikus užduotį galimybė taip pat nėra šiame tyrime etape aktuali, todėl nemodeliuojama ir neanalizuojama.




*Pav. 14. Gedimo scenarijus modeliuojantis šablonas*

### 5.1.3 RobotScout

Šablonas, kuriuo yra modeliuojamas robotas-žvalgas (žr. *Pav. 15*). Pradinė padėtis *begin* su eksponentiniu koeficientu *normRate* yra paliekama ir yra pereinama į laukimo padėtį *idle* egzistuojant sugedusiai valdymo stočiai ir/ar robotui-darbininkui. Iš *idle* pereinama į atitinkamą padėtį (*checkStation* arba *checkWorker*). Jeigu tuo metu nėra nei vieno laisvo roboto-mechaniko (patikrinimas *existsFreeMechanic*) sugrįžtama į laukimo būseną be jokių papildomų veiksmų. Jeigu analizuotas robotas yra sugedęs, pereinant į *idle* būseną yra atnaujinama roboto-mechaniko mechaniko informacija (*assignStationToMechanic* arba *assignWorkerToMechanic* atitinkamai) su roboto identifikaciniu numeriu. Gavus pranešimą iš išjungimo mechanizmo pasiekiama būsena *end*, skautas baigia darbą.

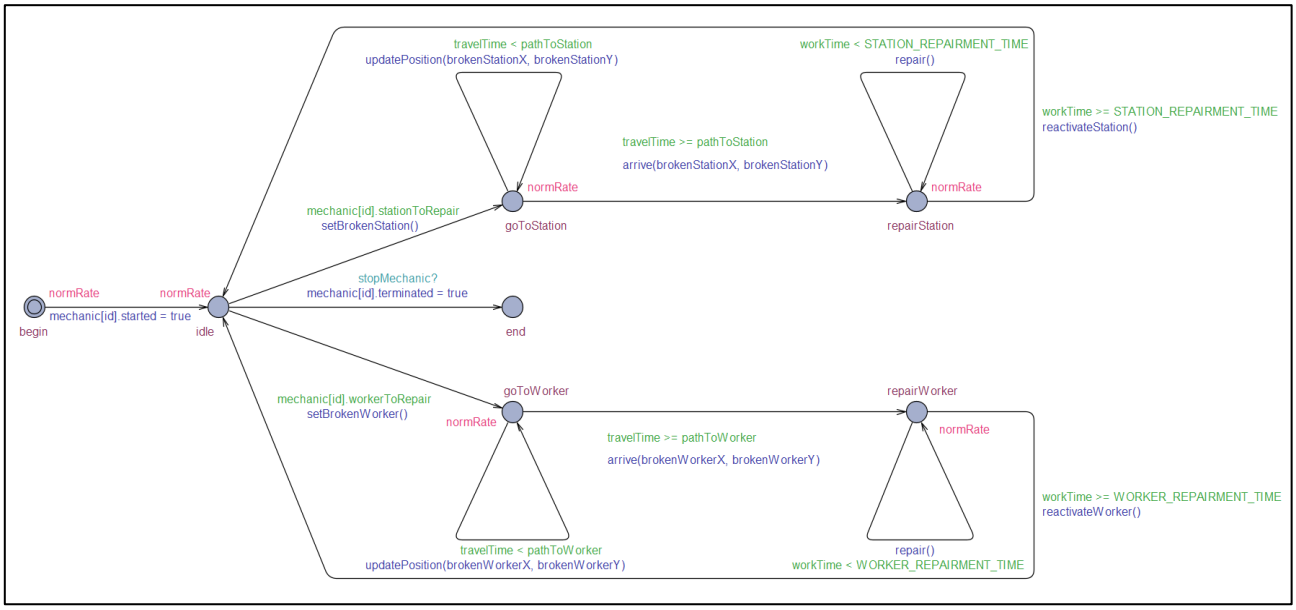



*Pav. 15. Robotą-žvalgą modeliuojantis šablonas*

### 5.1.4 RobotMechanic

Robotas-mechanikas yra modeliuojamas šablonu *RobotMechanic* (žr. *Pav. 16*). Pradinė padėtis yra paliekama su eksponentiniu koeficientu *normRate* ir pereinama į būseną *idle*. Priklausomai nuo to, kokia sąlyga yra tenkinama, pereinama į vieną iš trijų padėčių:

- Jeigu robotui-mechanikui yra priskirta valdymo stotis, iš kompiuterinės sistemos duomenų bazės yra užkraunama sugedusios komponentės informacija (*setBrokenStation*; nuskaitoma valdymo stoties lokalizacija bei apskaičiuojamas kelias iki jos; *travelTime* kintamasis prilyginamas nuliui – jis bus reikalingas siekiant simuliuoti kelią link stoties); padėtis *goToStation* yra paliekama robotui-mechanikui pasiekus sugedusią valdymo stotį (*travelTime* prilygus arba viršijus apskaičiuotą kelio laiką). Kintamasis, žymintis atliktą darbą – *workTime* – yra prilyginamas nuliui – dabar jis žymės laiką, kiek robotas-mechanikas taiso stotį. Kintamajam pasiekus apibrėžtą laiką (*STATION_REPAIRMENT_TIME*), valdymo stotis yra sutaisoma; grįžtama į padėtį *idle*;
- Jeigu robotui-mechanikui yra priskirtas robotas-darbininkas, yra atliekami identiški valdymo stoties taisymo atveju veiksmai sugedusio roboto-darbininko atžvilgiu;
- Gavus išjungimo mechanizmo pranešimą (*stopMechanic?*), pereinama į *end* padėtį – robotas-mechanikas užbaigia darbą.




*Pav. 16. Robotą-mechaniką modeliuojantis šablonas*

### 5.1.5 RobotWorker

Šablonas, kuriuo yra modeliuojamas robotą-darbininką atitinkantis procesas (žr. *Pav. 17*). Pradedant padėtimi *begin* su eksponentiniu koeficientu *normRate* yra pereinama į padėtį *idle*. Laukimo būsena gali būti palikta trimis atvejais:

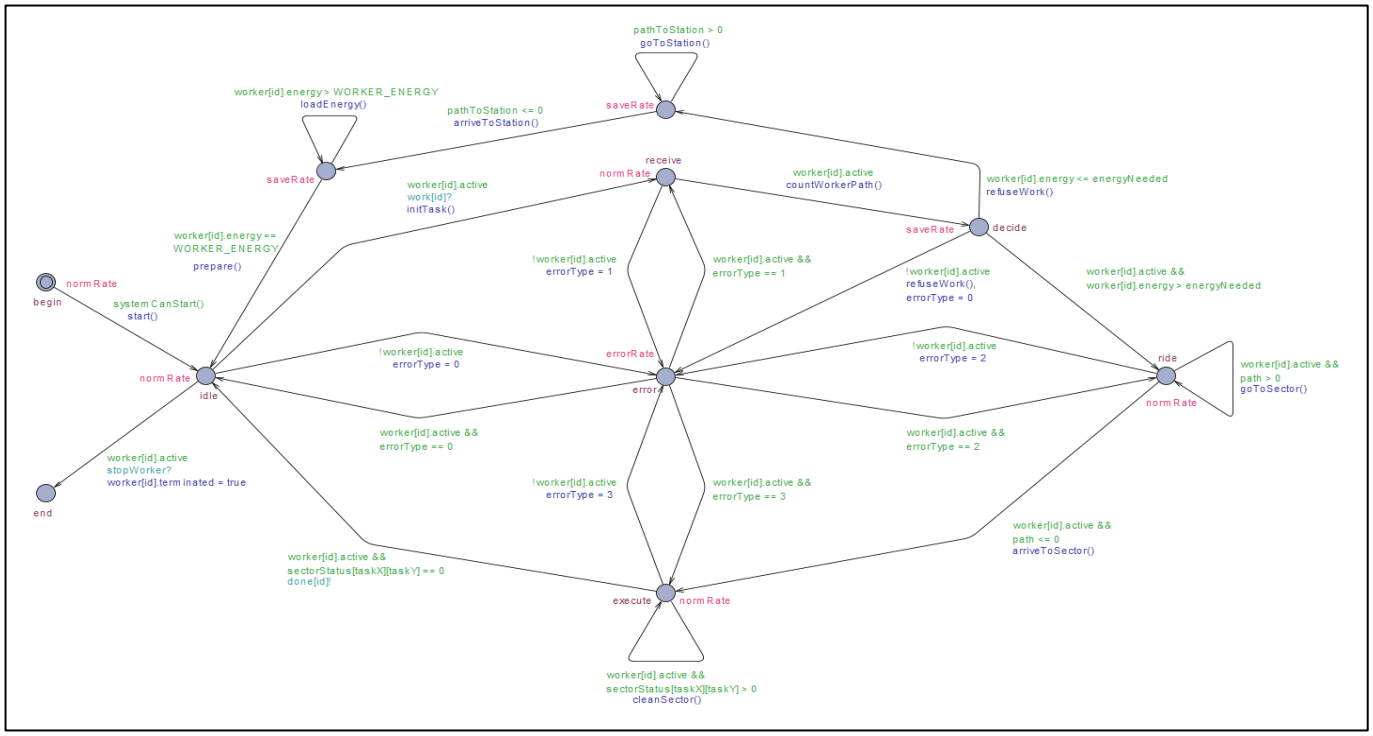
- Robotui-darbininkui būnant aktyviam (nesugedusiam) ir gavus pranešimą (*work*) iš valdymo stoties, kuriai robotas yra priskirtas. Robotas-darbininkas surenka visą reikiamą informaciją (metodas *initTask*; nuskaitomos priskirto sektoriaus koordinatės) bei patvirtina valdymo stočiai gavęs sektorių; pasiekia padėtį *receive*;
- Robotui-darbininkui būnant neaktyviam (sugedusiam) pereinama į klaidos padėtį *error* (išsaugomas klaidos tipas *errorType* – realiame gyvenime to robotas neatlieka; tai yra reikalinga modeliuojant, siekiant išsaugoti, kurioje būsenoje robotas buvo kai sugedo; šiuo atveju tai yra 0). Iš klaidos būsenos robotas grįžta į *idle* kai yra sutaisomas (jei *errorType* yra lygus 0);
- Robotui-darbininkui būnant aktyviam ir gavus išjungimo mechanizmo pranešimą apie darbo pabaigą – pereinama į padėtį *end*.

Padėtis *receive* yra paliekama jei robotas-darbininkas išlieka nesugedęs – apskaičiuojamas kelias iki priskirto sektoriaus. Jei pasiekęs padėtį robotas sugenda, pereinama į klaidos būseną (*errorType* priskiriama reikšmė 1).

Padėtyje *decide* robotas-darbininkas palygindamas energijos kiekius – šiuo metu turimos ir reikiamos atlikti priskirtai užduočiai – priima sprendimą atlikti užduotį ar jos atsisakyti ir paprašyti

maksimalaus įkrovimo (bei su eksponentiniu koeficientu *saveRate* grįžti į *idle* padėtį; R18). Skaičiuojant kiek energijos liktų pasiekus priskirtą sektorių ir jį išvalius (daroma prielaida kad nepasiekęs sektoriaus robotas-darbininkas negali iš anksto įvertinti, kiek laiko užims valymo darbai) naudojama maksimali sektoriaus nešvarumo reikšmė *MAX_DIRT*, todėl taip pat daroma prielaida, kad priėmus sprendimą sektorių valyti robotui-darbininkui energijos užteks. Tobulinant sistemą, šios prielaidos galima atsisakyti ir modeliuoti atvejus, kai dėl aplinkos pokyčių ar klaidos vertinime robotui-darbininkui neužteks energijos pilnam užduoties atlikimui.

Vertinimo metu robotas-darbininkas taip pat gali sugesti – tada darbo jis taip pat netenka; *errorType* priskiriama reikšmė 0 – sutaisytas robotas-darbininkas sugrįš į laukimo padėtį *idle*. Robotui-darbininkui būnant aktyviam ir priėmus sprendimą sektorių valyti pereinama į *ride* padėtį. Remiantis laikmačiu *workTime* modeliuojamas kelias link sektoriaus, kol galų gale jis yra pasiekiamas – pereinama į *execute* padėtį. Kelionės metu robotas-darbininkas gali sugesti bet kurio metu – ir tada pereinama į klaidos būseną (*errorType* lygu 2). Pasiekus *execute* padėtį pradedami valymo darbai – kol kintamasis, žymintis atliktą darbą, nėra lygus reikšmei nurodančiai sektoriaus nešvarumo lygį, robotas-darbininkas valo sektorių. Taip pat, bet kurio metu jis gali sugesti ir pereiti į būseną *error* (*errorType* = 3). Išvalęs sektorių aktyvus robotas-darbininkas pereina į padėtį *idle* siųsdamas pranešimą su juo susietai valdymo stočiai (*done[id]!*).




*Pav. 17. Robotą-darbininką modeliuojantis šablonas*

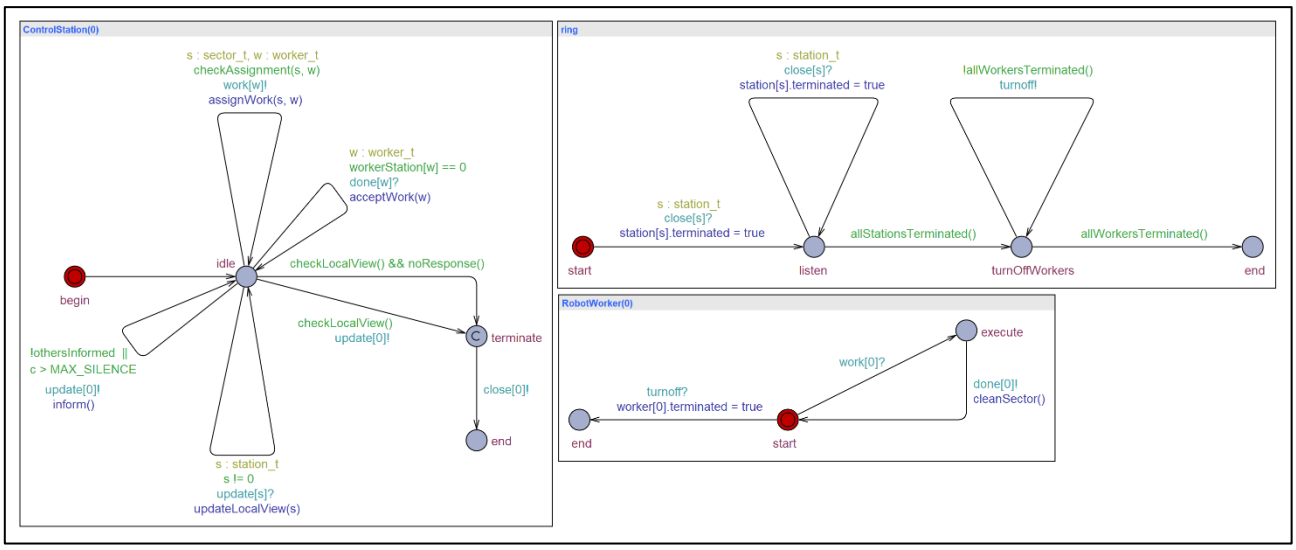
## 5.2 Formalus verifikavimas

Formalaus verifikavimo tyrimo dalį galima išskaidyti į dvi dalis – atliktą taikant klasikinį modelių patikrinimo metodą ir statistinį tos paties metodo plėtinį. Pirmoje tyrimo dalyje, verifikuojant pradinę sistemos versiją, buvo siekiama įrodyti jog adaptyvios kompiuterinės sistemos modelio pagrindas atitinka sistemai keliamus reikalavimus – visada sėkmingai užbaigia darbą atitinkamoje būsenoje. Pradinis modelis buvo tobulinamas bei verifikuojamas taikant statistinį modelių patikrinimą siekiant patyrinėti tikimybinius sistemos savybių įvertinimus.

### 5.2.1 Pradinė sistemos versija

Pirmoje adaptyvios kompiuterinės sistemos formalaus verifikavimo dalyje buvo modeliuojama pradinė sistemos versija (žr. *Pav. 18*). Siekiant maksimaliai patyrinėti pradinį sistemos modelį, buvo verifikuojamos įvairios struktūros:

- 2 valdymo stotys, 4 robotai-darbininkai, 1 skambučio mechanizmas, 9 sektoriai;
- 3 valdymo stotys, 6 robotai-darbininkai, 1 skambučio mechanizmas, 6 sektoriai;
- 4 valdymo stotys, 4 robotai-darbininkai, 1 skambučio mechanizmas, 6 sektoriai;




*Pav. 18. Pradinės kompiuterinės sistemos versijos konfigūracija*

Modelio patikrinimo metu, naudojant simuliatoriaus įrankį, buvo pastebėta, jog esant daugiau nei 2 valdymo stotims egzistuoja galimybė joms pasidalinti savo lokaliais vaizdais tokia tvarka, jog nepaisant visų sektorių išvalymo, nei viena valdymo stotis pagal savo lokalų vaizdą nelaikytų teritorijos pilnai išvalytos. Ši problema buvo išspręsta valdymo stotyje integravus laikmatį, skaičiuojantį praėjusį nuo paskutinio lokalaus vaizdo pasidalinimo su kita valdymo stotimi laiką. Kai nuo paskutinio laiko kitos valdymo stoties informavimo praėjo daugiau laiko vienetų už nurodytą

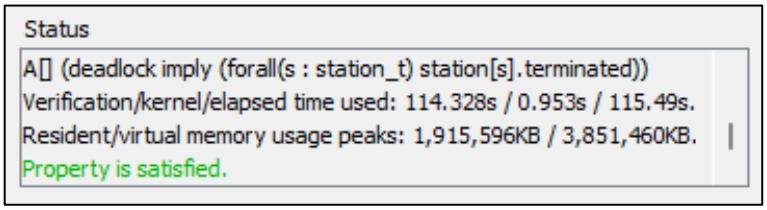
skaičių parametre *MAX_SILENCE*, aktyvi valdymo stotis taip pat gali dalintis savo lokaliu vaizdu, nepaisant to jog paskutiniu metu jai priklausantys sektoriai nepasikeitė.

#### 5.2.1.1 Pilno darbo atlikimo patikrinimas

Taikant modelių patikrinimo metodą, pradinė adaptyvios kompiuterinės sistemos modelio versija buvo formaliai verifikuojama. Siekiama įrodyti savybė buvo apibrėžta:

**A[] (deadlock imply (forall(s : station_t) ControlStation[s].end))**

„Visada aklavietės pasiekimas reiškia abiejų valdymo stočių buvimą darbo užbaigimo būsenoje“. Savybės patenkinimas buvo sėkmingai įrodytas (žr. *Pav. 19*), kas leidžia teigti, jog sistema nustoja veikusi (nebeatlieka jokių veiksmų) tik tada, kai pereina į darbo užbaigimo būseną.



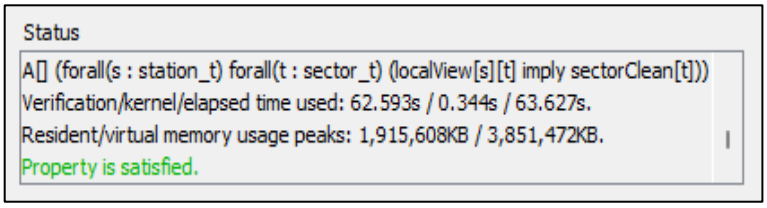

*Pav. 19. Pilno darbo atlikimo savybės įrodymo rezultatai*

#### 5.2.1.2 Valdymo stočių lokalių vaizdų korektiškumo patikrinimas

Ypatingai svarbia analizuojamos adaptyvios kompiuterinės sistemos savybe taip pat yra duomenų integralumas. Korektiško sistemos darbo reikalavimą galima išreikšti taisykle, jog valdymo stočių lokalaus vaizdas būtinai turi atitikti realią situaciją:

**A[] (forall(s : station_t) forall(t : sector_t) (localView[s][t] imply sectorClean[t]))**

„Visada, jei pagal bet kurios valdymo stoties lokalų įsivaizdavimą pasirinktas sektorius yra išvalytas, tas sektorius iš tikrųjų turi būti išvalytas“. Situacija, kai pagal kurios nors valdymo stoties lokalų vaizdą sektorius yra išvalytas, nors iš tikrųjų toks nėra, yra laikoma grubia ir fundamentalia sistemos klaida. Sėkmingai įrodžius savybę (žr. *Pav. 20*) galima teigti, jog pirminis adaptyvios kompiuterinės sistemos modelis veikia korektiškai.




*Pav. 20. Lokalaus valdymo stoties vaizdo korektiškumo įrodymo rezultatai*

### 5.2.2 Pilna sistemos versija

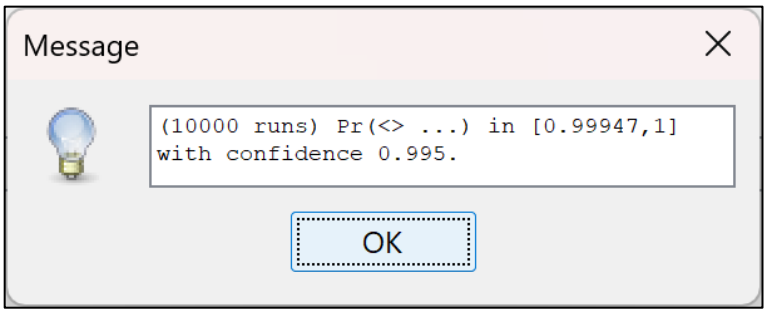
Įrodžius pradinės sistemos modelio korektiškumą, modelis buvo tobulinamas. Kiekviename iš egzistavusių šablonų buvo pridėtos naujos padėtys ir perėjimai tarp jų; pvz. klaidos būsena *ControlStation* šablone, į kurią patenkama jei valdymo stotis sugenda. Buvo pridėti nauji šablonai, atitinkantys roboto-mechaniko bei roboto-skauto procesus, gedimų modeliavimą ir resursų valdymo programą. Sistemos konfigūracija buvo atnaujinta; ją nuo šiol sudaro 4 valdymo stotys, 12 robotų-darbininkų, 25 sektoriai, po vieną robotą-mechaniką ir robotą-skautą.

#### 5.2.2.1 Tikimybinis sistemos pilno darbo atlikimo įvertinimas

Naudojant UPPAAL SMC buvo siekiama įvertinti, su kokia tikimybe naujoje adaptyvios kompiuterinės sistemos modelio versijoje (konfigūracijoje) darbas bus užbaigtas pilnai atlikus užduotį (išvalius visą teritoriją) – t.y. kokia tikimybė, jog po 200 laiko vienetų visų sektorių reikšmės masyve *sectorStatus* taps lygios nuliui*:*

**Pr[<=200; 10000]( <> forall(t : territory_t) sectorIsCleaned(t) )**

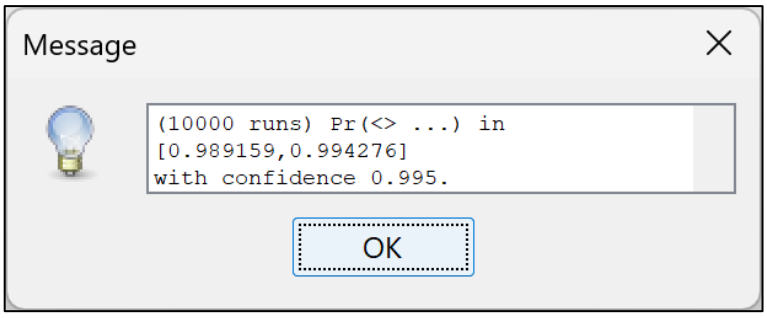
Iš pradžių buvo atliktas eksperimentas be gedimo proceso *DeactivationProcess,* siekiant sužinoti, kiek laiko vienetų pakanka norint teigti jog kompiuterinė sistema be gedimų pilnai atliks darbą. Buvo patikrinti 10 000 skirtingų sistemos darbo vykdymo scenarijų – kiekviename iš jų kažkurio metu savybė turėjo būti patenkinta. Teigiamas rezultatas (savybės tenkinimas) buvo pasiektas kiekviename iš bandymų – tikimybė, jog negendanti kompiuterinė sistema pilnai išvalo visą teritoriją per 200 laiko vienetų siekia 1 (žr. *Pav. 21*). Vidutiniškai sistemai pakakdavo apie 90 laiko vienetų.




*Pav. 21. Tikimybinio sistemos pilno darbo atlikimo įvertinimo rezultatai (200 laiko vienetų, sistemos komponentės negenda)*

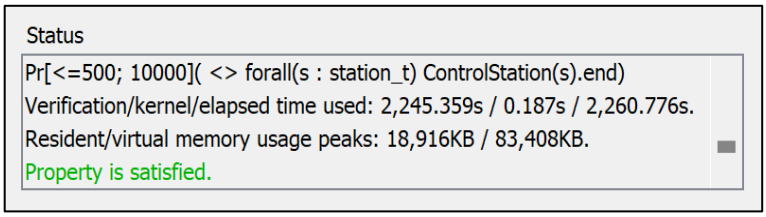
Eksperimentas buvo tęsiamas pridėjus robotų gedimo galimybę. Tai turi prailginti sistemos darbo laiką, todėl tikrinama, ar sistema, kurios valdymo stotys ir robotai-darbininkai gali sugesti, pilnai atlieka darbą per 500 laiko vienetų.

Gauti rezultatai kiek prastesni už ankstesnius (žr. *Pav. 22*). Tikimybė, jog sistema, kurią sudarantys valdymo stotys ir robotai-darbininkai gali sugesti, pilnai išvalo jai priskirtą teritoriją, yra apibrėžiama intervale nuo 0,989159 iki 0,994279.



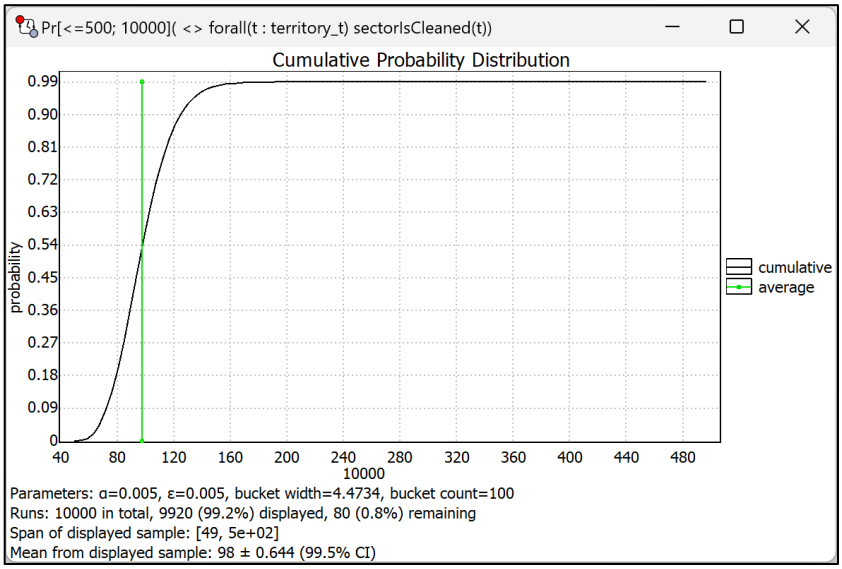

*Pav. 22. Tikimybinio sistemos pilno darbo atlikimo įvertinimo rezultatai (500 laiko vienetų, sistemos komponentės gali sugesti)*

Tai, jog viršutinis tikimybės rėžis nesiekia 1 (kas reikštų jog per nurodytą laiko tarpą kiekvieno patikrinimo metu sąlyga buvo įvykdyta), paaiškinamas faktu, jog 10000 patikrinimų metu buvo rasti vykdymo scenarijai, trukę ilgiau nei 500 laiko vienetų – toks scenarijus įmanomas pvz. valdymo stotims nuolat gendant. Pasiekta tikimybė yra pakankama, kad galima būtų teigti jog savybė yra patenkinama (žr. *Pav. 23*).



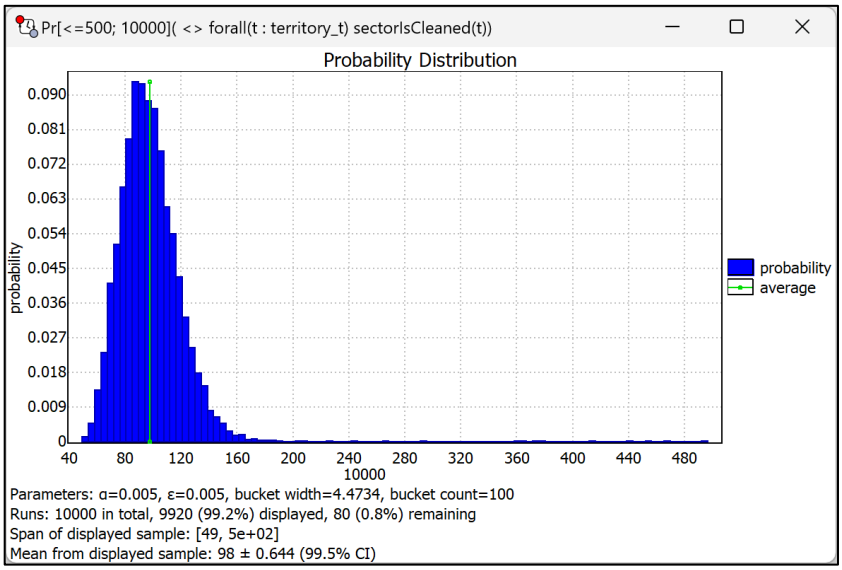

*Pav. 23. Tikimybinio sistemos pilno darbo atlikimo įvertinimo statusas (500 laiko vienetų, sistemos komponentės gali sugesti)*

UPPAAL SMC įrankio pagalba atliktus statistinius modelių patikrinimus galima vizualizuoti (žr. *Pav. 24*). Iš sugeneruotos kumuliatyvaus tikimybių pasiskirstymo diagramos galima pastebėti, jog ties 160 laiko vienetu pasiekiama 0,99 sėkmingo darbo atlikimo tikimybė, kuri nežymiai didėja ilgėjant laiko intervalui. Sąlyginai greitas plynaukštės (ang. *plateau*) – taško, nuo kurio tikimybė didėja nežymiai – pasiekimas yra stabilios (prognozuojamos, neturinčios netikėtų funkcionalumo sutrikimų) sistemos požymis.




*Pav. 24. Pilno sistemos darbo atlikimo kumuliatyvaus tikimybių pasiskirstymo diagrama*

Kitos diagramos – vaizduojančios tikimybių pasiskirstymą (žr. *Pav. 25*) – dėka galima pastebėti, jog su tikimybe 0,18 sistema užbaigs darbą ties 90-95 laiko vienetais. Bendru atveju, analizuojamam modeliui turint vieną, su didele tikimybe pasikartojančią, tikrinamą savybę tenkinančią reikšmę, modelis taip yra laikomas stabilus. Tikimybių paskirstymo diagramoje esant keliems pikams (aukščiausiems taškams) arba kumuliatyvaus tikimybių paskirstymo diagramoje, po ganėtinai lėto kitimo, esant smarkiam tikimybės šuoliui, galima teigti apie analizuojamo objekto (sistemos) nestabilumą, kas yra traktuojama kaip objekto trūkumas.



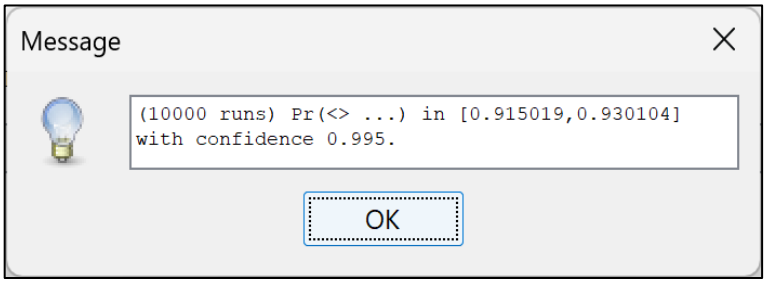

*Pav. 25. Pilno sistemos darbo atlikimo tikimybių pasiskirstymo diagrama*

#### 5.2.2.2 Tikimybinis visų valdymo stočių darbo pabaigimo įvertinimas

Išsiaiškinus, jog adaptyviai kompiuterinei sistemai pakanka apie 500 laiko vienetų tam, kad būtų užtikrintas visų sektorių išvalymas, buvo tyrinėjama, kiek laiko prireikia valdymo stotims apsikeisti lokaliais vaizdais bei priimti sprendimą pabaigti darbą. Suformuluota savybė, jog per 500 laiko vienetų kažkuriuo metu visos valdymo stotys pasiekia darbo pabaigos būseną *end*:

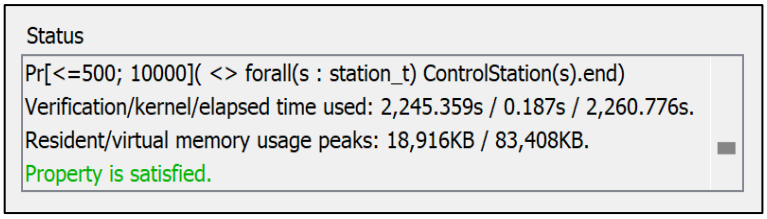
**Pr[<=500; 10000]( <> forall(s : station_t) ControlStation(s).end)**

Tikimybė, jog praėjus 500 laiko vienetų nuo sistemos darbo pradžios visos valdymo stotys bus darbo užbaigimo būsenoje *end* buvo įvertinta tarp 0,915019 ir 0,930104 (žr. *Pav. 26*). Lyginant su rezultatais, gautais tiriant pilno darbo atlikimo tikimybę (skyrelis 5.2.2.1), pastebimas suprastėjimas (apie 0,065). Nepaisant to, jog sistema praktiškai visada pilnai išvalo teritoriją per 500 laiko vienetų, valdymo stotys per tiek pat laiko tai identifikuoja bei užbaigia darbus tik apie 9 atvejais 10.



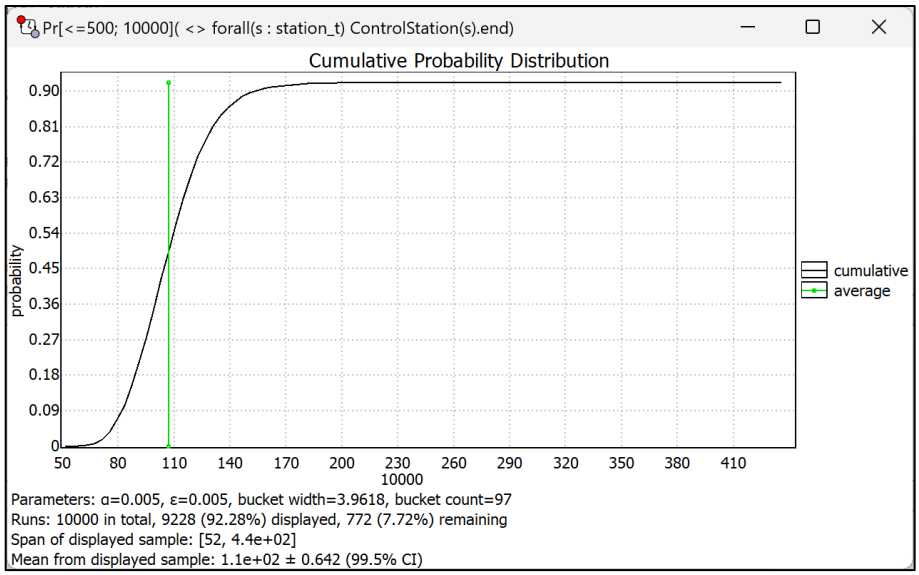

*Pav. 26. Tikimybinio visų valdymo stočių darbo pabaigimo įvertinimo rezultatai*

Tikimybės, jog savybė yra tenkinama per nurodytą laikotarpį, suprastėjimą galima paaiškinti valdymo stočių gedimais jau faktiškai išvalius arba priskyrus robotams-darbininkams visus su jomis susietus sektorius (besidalinant savo lokaliais vaizdais ir belaukiant jų iš kitų valdymo stočių) bei komunikacijos trikdžiais (nepaisant dalinimusi lokaliu vaizdu, jokia kita valdymo stotis pranešimo negauna). Pasiekta tikimybė yra pakankama, kad galima būtų teigti jog savybė yra patenkinama (žr. *Pav. 27*).




*Pav. 27. Tikimybinio visų valdymo stočių darbo pabaigimo įvertinimo statusas*

Analizuojant eksperimento kumuliatyvaus tikimybių pasiskirstymo diagramą (žr. *Pav. 28*) pastebimas ir vidutinio laiko savybei patenkinti padidėjimas. Jeigu pilnam teritorijos išvalymui vidutiniškai pakanka 98 laiko vienetų, tai valdymo stotims vidutiniškai užtrunka pabaigti darbą apie 110 laiko vienetų.



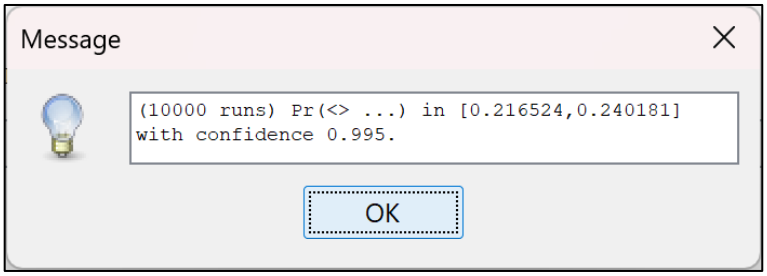

*Pav. 28. Visų valdymo stočių darbo pabaigimo kumuliatyvaus tikimybių pasiskirstymo diagrama*

### 5.2.2.3 Tikimybinis valdymo stočių gedimo galimybės įvertinimas

Tęsiant eksperimentus buvo siekiama tyrinėti kitas sistemos savybes. Vienu iš tyrimo tikslų buvo pasirinkta tikimybė, jog per pasirinktą laiko tarpą bent viena iš valdymo stočių suges. Užklausa:

**Pr[<=500; 10000] (<> exists(s : station_t) ControlStation(s).error)**

leidžia įvertinti, su kokia tikimybe per 500 laiko vienetų bet kuri valdymo stotis bent kartą pasieks klaidos būseną. Atliktas 10000 sistemos veikimo scenarijų patikrinimas (žr. *Pav. 29*) leidžia teigti, kad tikimybė yra pakankamai didelė – tarp 0,216524 ir 0,240181.



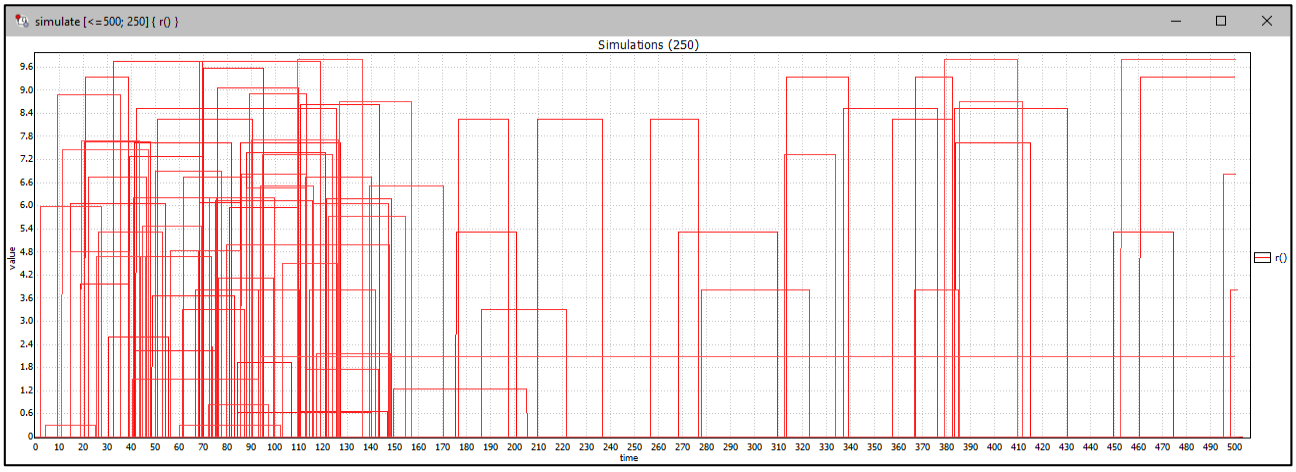

*Pav. 29 Tikimybinio valdymo stočių gedimo galimybės įvertinimo rezultatai*

#### 5.2.2.4 Valdymo stočių gedimo laikotarpio simuliavimas

Išsiaiškinus, jog valdymo stoties gedimo galimybė yra pakankamai reali, buvo siekiama patyrinėti, kiek laiko vidutiniškai sistema veikia su bent viena sugedusia valdymo stotimi. Tikrinant užklausą:

**simulate [<=500; 250] { exists(s : station_t) ControlStation(s).error }**

simuliuojami 250 sistemos veikimo scenarijų, trunkančių iki 500 laiko vienetų. Gauti rezultatai (žr. *Pav. 30*) iliustruoja valdymo stočių gedimo atsitiktinumą. Esti atvejų, kai valdymo stotys pradeda gesti ties 200 laiko vienetų riba ir genda paeiliui (pagal vertikalią grafiko reikšmę 8.2), tačiau būna atvejų, kai sistema dirba bent vienos valdymo stoties gedimo būsenoje daugiau nei 350 laiko vienetų (vertikali grafiko reikšmė 2.2) – tai nebūtinai reiškia, jog visą laiką buvo sugedusi ta pati valdymo stotis, neveiksniomis galėjo būti kelios valdymo stotys vienu metu. Bendru atveju, laikotarpis kai bent viena valdymo stotis yra sugedusi trunka tarp 30 ir 40 laiko vienetų – turint omenyje, jog parametrui *STATION_REPAIRMENT_TIME*, nurodančiam kiek laiko vienetų robotas-mechanikas turi taisyti sugedusią stotį, buvo priskirta reikšmė 20, patį gedimo faktą turi pastebėti robotas-skautas, pranešti apie tai robotui-mechanikui ir robotui-mechanikui dažniausiai tenka bent kelis laiko vienetus keliauti link stoties, sistemos reakcijos laiką į gedimą galima laikyti pakankamai greitu.




*Pav. 30. Valdymo stočių gedimo laikotarpio simuliavimo rezultatai*

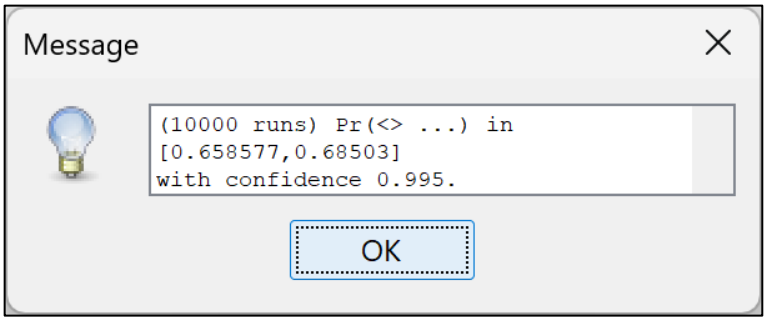
Siekiant optimizuoti adaptyvią kompiuterinę sistemą ir jos modelį, robotui-skautui arba robotui-mechanikui galima apibrėžti valdymo stoties prioretizavimo principą – vienu metu esant sugedusiems valdymo stočiai ir robotui-darbininkui, taisymo pirmenybė būtų teikiama valdymo stočiai.

#### 5.2.2.5 Tikimybinis robotų-darbininkų gedimo galimybės įvertinimas

Panašiai eksperimentui, aprašytame 5.2.2.3 skyrelyje, buvo siekiama įvertinti tikimybę, jog sistemos veikimo metu suges bent vienas robotas-darbininkas. Pritaikius užklausą robotams-darbininkams:

**Pr[<=500; 10000] (<> exists(w : worker_t) RobotWorker(w).error)**

galima atlikti tikimybinį tokio įvykio įvertinimą. Kadangi robotų-darbininkų yra triskart daugiau nei valdymo stočių, o pagal nurodytus *DeactivationProcess* procesui tikimybinius svorius roboto-darbininko gedimo galimybė yra labiau tikėtina, gauti rezultatai patvirtina hipotezę, jog ši tikimybė turi būti aukštesnė už valdymo stočių gedimo tikimybę (žr. *Pav. 31*). Tikimybė tarp 0,658577 ir 0,68503 žymi, kad praktiškai dviejuose iš trijų teritorijos valymų bent viena sistemos komponentė genda.



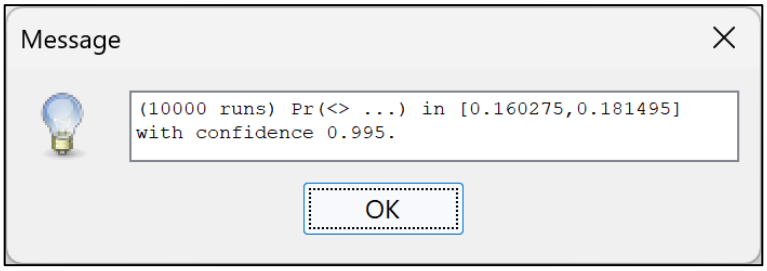

*Pav. 31. Tikimybinio robotų-darbininkų gedimo galimybės įvertinimo rezultatai*

#### 5.2.2.6 Tikimybinis kelių robotų-darbininkų gedimo galimybės įvertinimas

Robotų-darbininkų gedimo galimybės analizė yra tęsiama – siekiant įvertinti tikimybę, jog kažkuriuo sistemos darbo metu bent du robotai-darbininkai lygiagrečiai bus sugedę:

**Pr[<=1000; 10000] (<> (exists(w1 : worker_t) exists(w2 : worker_t) RobotWorker(w1).error && RobotWorker(w2).error && w1 != w2) )**

Gautus rezultatus (žr. *Pav. 32*) lyginant su gautais praeito eksperimento metu, galima pastebėti, jog tikimybė sumažėja apie 4 kartus. Svarbu pabrėžti, jog 5.2.2.5 eksperimento metu yra tikrinama ar bent vienas robotas-darbininkas yra sugedęs (yra skaičiuojami ir atvejai kai jų yra daugiau); šiame skyrelyje aprašomas patikrinimas įtraukia atvejus, kai sugedusių robotų-darbininkų yra du ir daugiau.



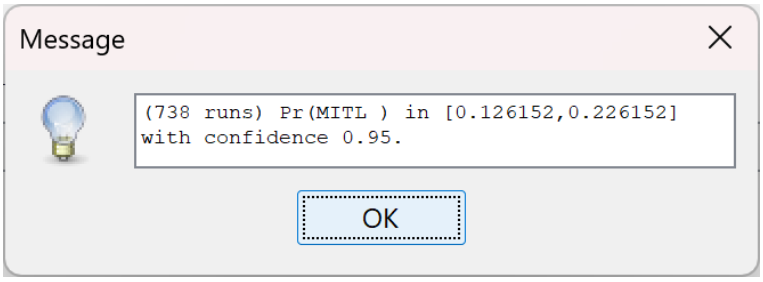

*Pav. 32. Tikimybinio kelių robotų-darbininkų gedimo galimybės įvertinimo rezultatai*

5.2.2.7 Tikimybinis robotų-darbininkų gedimo per nurodytą laiko tarpą galimybės įvertinimas

Alternatyviai valdymo stočių gedimo simuliacijai, aprašytai 5.2.2.4 skyrelyje, robotų gedimo laikas gali būti analizuojamas tikimybiniais įverčiais – tačiau tokiu atveju laiką, kurį robotas praleidžia gedimo būsenoje, reikia apibrėžti. Roboto-darbininko taisymo laikas (parametras *WORKER_REPAIRMENT_TIME*) buvo lygus 10 laiko vienetų. Nurodomas laiko tarpas yra 30 vienetų turint omenyje, jog bendru atveju praeis kiek laiko, kol roboto-darbininko gedimą pastebės robotas-skautas, o robotas-mechanikas pasieks sugedusią komponentę. Užklausa:

**Pr( <>[0, 500] ([] [0, 30] (exists(w : worker_t) RobotWorker(w).error)))**

leidžia gauti tikimybinį įvertinimą, jog per pirmus 500 laiko vienetų bent vienas robotas-darbininkas 30 laiko vienetų iš eilės bus gedimo būsenoje. Rezultatų (žr. *Pav. 33*) pagrindu galima teigti, jog ilgesnis roboto-darbininko gedimas yra pastebimai mažiau tikėtinas už paties gedimo tikimybę, kas implikuoja optimalų roboto-skauto ir roboto-mechaniko darbą.




*Pav. 33. Tikimybinio robotų-darbininkų gedimo per nurodytą laiko tarpą galimybės įvertinimo rezultatai*

5.2.2.8 Robotų-darbininkų energijos sunaudojimo simuliavimas

Tyrimas buvo pratęstas siekiant patyrinėti, kaip laiko perspektyvoje keičiasi robotų-darbininkų turimas energijos kiekis. Robotai pradėdavo darbą turėdami maksimalų baterijos įkrovimą (parametras *ROBOT_ENERGY* – su priskirta reikšme 25, kadangi tiek energijos pakaktų vienam

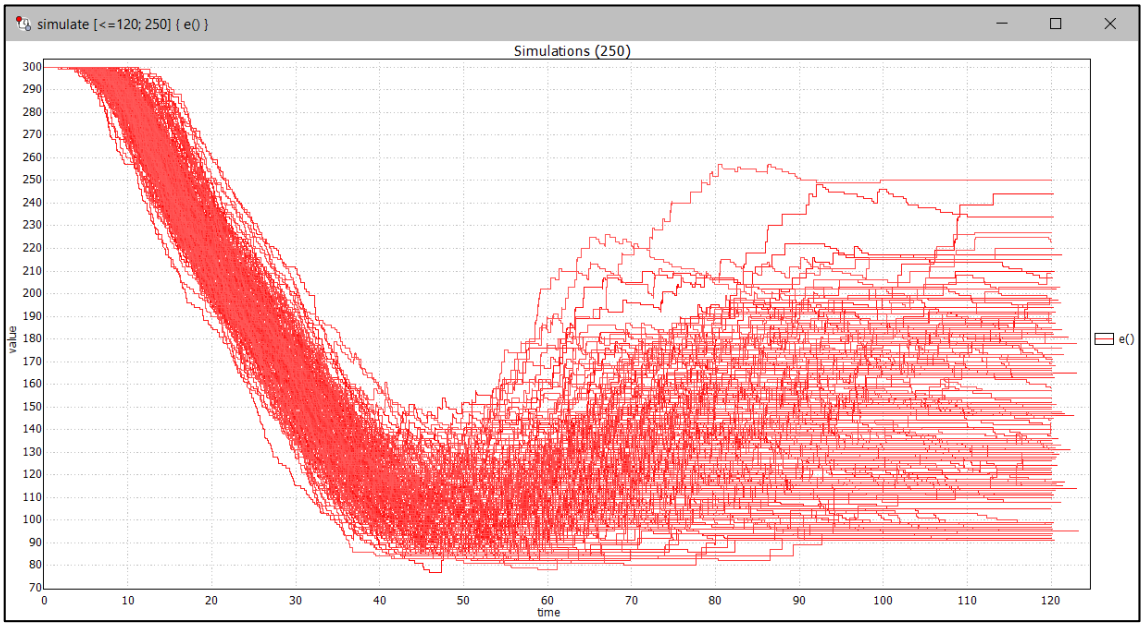
ilgiausiai trunkančiam sektoriaus valymui – turint omenyje atstumus tarp sektoriaus ir roboto-darbininko bei stoties ir maksimalų sektoriaus užterštumo lygį). Užklausa:

**simulate [<=120; 250] { sum (w : worker_t) worker[w].energy }**

Per 120 laiko vienetų (reikšmė pasirinkta turint omenyje 5.2.2.1 skyrelio eksperimento rezultatus, kurių pagrindu galima teigti, jog pilnam teritorijos išvalymui vidutiniškai pakanka apie 98 laiko vienetų), atliekant sistemos darbo simuliavimą 250 kartus, kiekvieną laiko vienetą skaičiuojama visų robotų-darbininkų baterijų įkrovimų suma.

Gauti rezultatai (žr. *Pav. 34*) iliustruoja tendenciją skirtinguose sistemos veikimo scenarijuose. Robotai-darbininkai pradeda eikvoti energiją (t.y. judėti link priskirto sektoriaus, pradėti valymo darbus) ties 5-15 laiko vienetų riba – tai galima paaiškinti tuo, jog sistemos veikimo pradžioje komponentės turi įsijungti, ir tik tada valdymo stotys pradeda skirti darbus robotams-darbininkams.

Energijos kiekis tolygiai mažėja (robotai-darbininkai dirba visą laiką) iki 30-40 laiko vienetų – ties šiuo momentu dauguma robotų maksimaliai išeikvoja bateriją, ir dėl to atsisako darbų ir vyksta valdymo stoties link, įkrovimui. Po 50 laiko vieneto skirtingi sistemos vykdymo scenarijai išsiskiria – dėl gedimų ir priskiriamų sektorių eiliškumo.



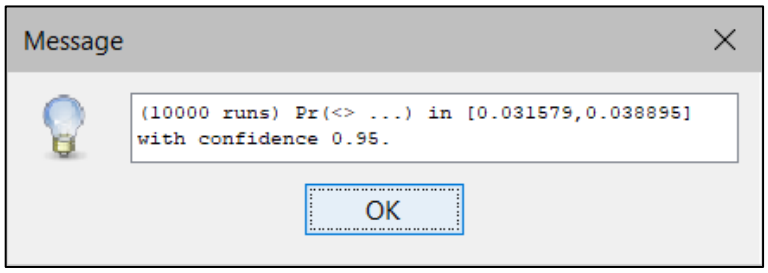

*Pav. 34. Robotų-darbininkų energijos sunaudojimo simuliavimo grafikas*

5.2.2.9 Tikimybinis valdymo stoties ir roboto-darbininko gedimo galimybės įvertinimas

Kadangi valdymo stočių ir robotų-darbininkų gedimų tikimybės yra pakankamai didelės, buvo pasiekiama patyrinėti, kokia yra tikimybė, jog kažkuriuo metu suges tiek valdymo stotis, tiek su ja susietas robotas-darbininkas. Suformuluotos užklausos:

**Pr[<=500; 10000] (<> exists exists(s : station_t) (w : worker_t) ControlStation(s).error && RobotWorker(w).error && worker[w].station == s)**

patikrinimo rezultatai nenurodo dažną tokios situacijos pasikartojimą (žr. *Pav. 35*). Turint omenyje, jog valdymo stotys gedimo būsenoje vidutiniškai būna apie 30-40 laiko vienetus (5.2.2.4 eksperimentas), o robotai-darbininkai – mažiau nei 30 (5.2.2.7), o ir patys robotai genda ne kiekvieno teritorijos valymo metu, minimali tikimybė jog tuo pačiu metu bus sugedusi ir valdymo stotis, ir jai priskirtas robotas, atrodo logiškai.




*Pav. 35. Tikimybinio valdymo stoties ir roboto-darbininko gedimo įvertinimo rezultatai*

5.2.2.10 Išvalytų sektorių per nurodytą laiko tarpą skaitinis įvertinimas

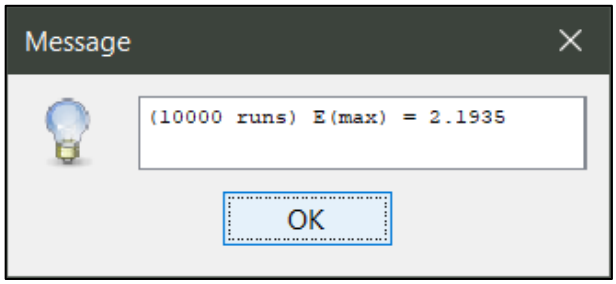
Antroji tyrimo dalis yra finalizuojama numatomos reikšmės maksimumo ir minimumo įvertinimo eksperimentais. Atliekant tyrimus bus siekiama išsiaiškinti, koks yra maksimalus ir minimalus išvalytų sektorių skaičius konkrečiu laiko momentu.

5.2.2.10.1 Maksimalus skaičius

Eksperimento, aprašyto 5.2.2.1. skyrelyje, rezultatai leidžia teigti, jog kompiuterinei sistemai vidutiniškai pakanka apie 98 laiko vienetų pilnai išvalyti jai priskirtą teritoriją. Ties apie 160 laiko vienetų riba pilno visų sektorių išvalymo tikimybė siekia 0,99. Siekiant patyrinėti, kiek maksimaliai sektorių gali išvalyti esamos konfigūracijos sistema per 15 pirmų laiko vienetų, buvo suformuluota užklausa:

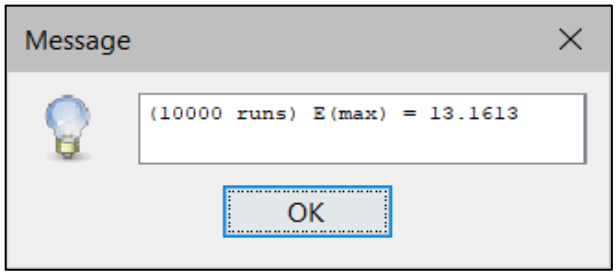
**E[<=15; 10000](max: sum(t : territory_t) sectorIsCleaned(t) )**

Gauti rezultatai (žr. *Pav. 36*) leidžia teigti, jog geriausiu atveju (sistemos komponentės nesugedo, valdymo stotys ir įsijungė maksimaliai greitai, robotai-darbininkai buvo arti priskirtų sektorių) per pirmų 15 laiko vienetų sistema gali išvalyti 2 sektorius.




*Pav. 36. Maksimalaus išvalytų per pirmus 15 laiko vienetų sektorių skaičiaus įvertinimo rezultatai*

Analizuojamą laikotarpį prailginus 2 kartus (30 laiko vienetų), maksimalus išvalytų sektorių skaičius padidėjo iki 13 (žr. *Pav. 37*), kas yra daugiau nei 50% visų teritoriją sudarančių sektorių skaičiaus. Vėlgi, svarbu paminėti, jog kadangi tai yra numatomos reikšmės maksimumas, konkrečiai tokį rezultatą pasiekęs sistemos darbo scenarijus buvo maksimaliai efektyvus ir sėkmingas (sistema nesusidūrė su gedimais).




*Pav. 37. Maksimalaus išvalytų per pirmus 30 laiko vienetų sektorių skaičiaus įvertinimo rezultatai*

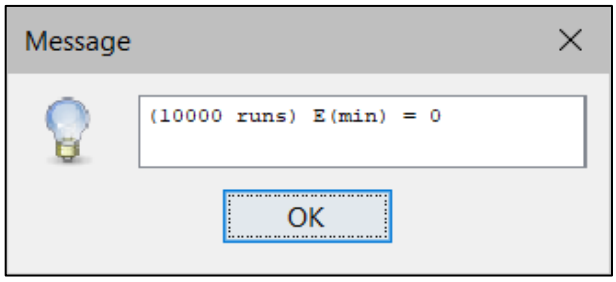
5.2.2.10.2 Minimalus skaičius

Analogiškai, buvo siekiama patyrinėti koks gali būti mažiausias išvalytas sektorių skaičius. Renkantis laiko vienetą, kurio metu bus skaičiuojamas numatomos reikšmės minimumas, buvo taikoma atvirkštinė maksimumo atvejui logika – jeigu ieškant didžiausios reikšmės buvo parenkami kuo mažesni laiko vienetai, siekiant patyrinėti kiek daugiausia idealiu atveju sistema išvalo sektorių, siekiant įvertinti minimalią reikšmę laikotarpį derėtų plėsti. Eksperimentu siekiama įvertinti, kiek minimaliai sektorių gali būti išvalyta ties 500-iu laiko vienetu:

**E[<=500; 10000](min: sum(t : territory_t) sectorIsCleaned(t) )**

Gautas rezultatas gali būti netikėtas (žr. *Pav. 38*), tačiau yra logiškas – kadangi sistemos modelyje nebuvo suformuluotos jokios prielaidos dėl robotų gedimo skaičiaus, teoriškai, įmanomas sistemos

veikimo scenarijus, kai valdymo stotys ir robotai-darbininkai ges nuolat – tokiu atveju, net ir po 500 laiko vienetų, įmanoma, jog nei vienas sektorius nebus pilnai išvalytas.




*Pav. 38. Minimalaus išvalytų per pirmus 500 laiko vienetų sektorių skaičiaus įvertinimo rezultatai*

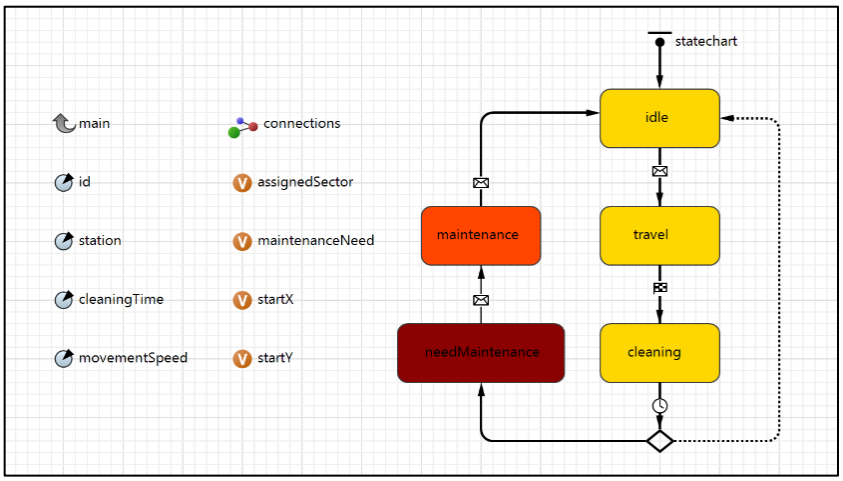
## 5.3 Simuliavimas

Statistinis modelių patikrinimas, leidžiantis formaliai verifikuoti sudėtingas sistemas, reikalauja aiškių analizės laiko rėžių. Kiekvienoje iš šiame darbe pristatytų UPPAAL SMC užklausų turėjo būti nurodytas laiko vienetas, iki kurio turėjo būti tikrinama apibrėžta savybė. Tikslių rėžių nustatymas gali būti naudingas jei siekiama analizuoti tik konkretų sistemos veikimo momentą arba laikotarpį, tačiau bendru atveju riboja verifikuojamo objekto analizę. Šio darbo tyrime analizuojama adaptyvi teritoriją valanti robotų sistemą turi jai priskirtą konkrečią užduotį, kurią atlikus (pasiekus norimą rezultatą), darbas yra baigiamas. Tačiau jeigu sistemos užduotis būtų ne išvalyti teritoriją, o palaikyti tvarką joje neapibrėžtą (ar maksimaliai įmanoma) laikotarpį, statistinis modelių patikrinimas kaip verifikavimo metodas nebebūtų optimalus.

Iš esmės, neapibrėžtą (potencialiai begalinį) tiriamojo objekto darbą galima formaliai verifikuoti taikant klasikinį modelių patikrinimą – tačiau jeigu verifikuojamas objektas yra per sudėtingas, dėl būsenų sprogimo gauti rezultatų nepavyks (šio darbo tyrimas iliustruoja tokį atvejį). Klasikinis modelių patikrinimas taip pat neturi galimybės gauti verifikuojamo objekto statistiką, numatomas reikšmes ir tikimybes, jog pasirinkta savybė bus patenkinta. Pilną potencialiai begalinį veikimo scenarijų turintį objektą formaliai verifikuoti galima taikant teoremų įrodymo metodą. Antra vertus, teoremų įrodytojai turi ribotą galimybę esant poreikiui dinamiškai patyrinėti kaip sistemos veikimo metu yra reaguojama į konkretų atvejį ar situaciją, pvz. kaip sistema reaguoja į pasirinktą parametrų pokytį.

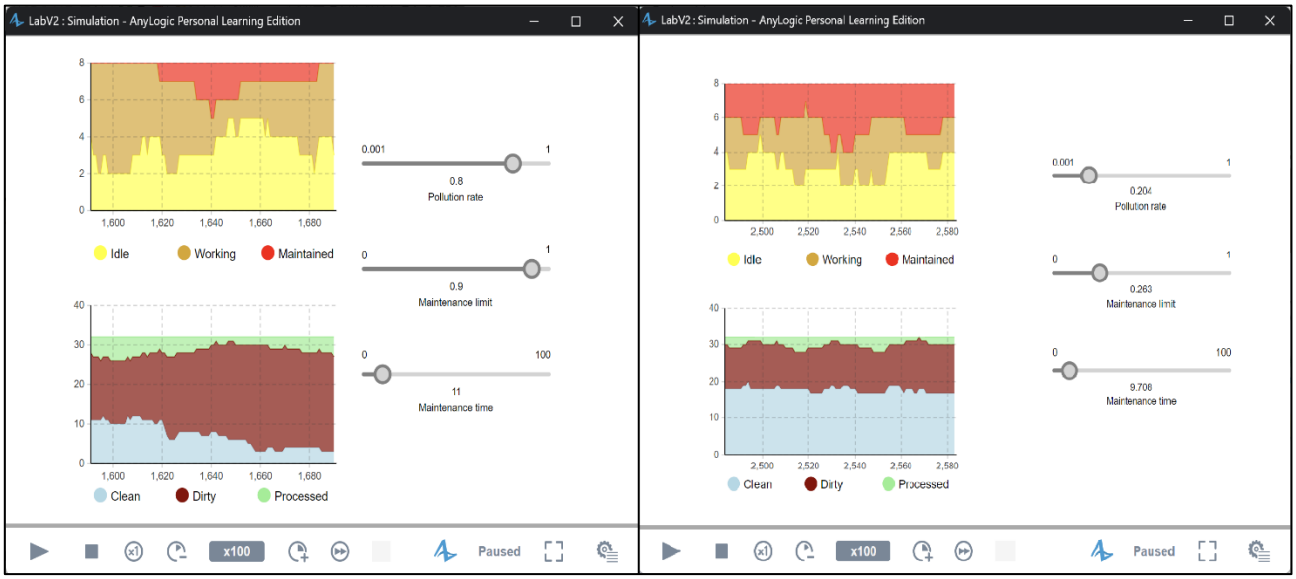
Detalią tiriamojo objekto analizę veikimo metu galima atlikti naudojant simuliavimą. Nors šis objekto tyrimo metodas negali būti laikomas lygiaverčiu formaliems metodams, jo dėka galima analizuoti kiekvieną konkretų atvejį atskirai bet generuoti įvairius siekiamus įvertinti scenarijus ir

būsenas. Lygiagrečiai pagrindinei, esminei tyrimo daliai – adaptyvios kompiuterinės sistemos formaliam verifikavimui taikant modelio patikrinimo metodą (bei statistinį modelio patikrinimo metodą) UPPAAL įrankio ir UPPAAL SMC plėtinio pagalba – buvo plėtojamas analogiškas modelis AnyLogic aplinkoje (žr. *Pav. 39*) [Any22]. Modelis, kurį sudaro visos aprašytos kompiuterinės sistemos komponentės (valdymo stotys, robotai-darbininkai, robotai-medikai ir robotai-skautai), lyginant su sukurtu naudojant UPPAAL įrankį, yra abstraktesnis.




*Pav. 39. Roboto-darbininko šablonas AnyLogic aplinkoje*

Papildant darbe pristatytą tyrimą, atliktą formaliai verifikuojant adaptyvią kompiuterinę sistemą modelių patikrinimo įrankiu UPPAAL, buvo atlikti simuliavimai AnyLogic aplinkoje (žr. *Pav. 40*).




*Pav. 40. Adaptyvios švarą teritorijoje palaikančios kompiuterinės robotų sistemos darbo simuliavimo AnyLogic aplinkoje rezultatai*

Simuliuojant sukurtą modelį, pilną teritorijos išvalymą, formaliai patikrintą UPPAAL dėka, pavyko patvirtinti. Modelis buvo praplėstas – sektoriams buvo suteikta galimybė po išvalymo kažkuriuo metu vėl tapti užterštais. Tokiu būdu iš esmės pasikeitė teritoriją valančios sistemos užduotis – vietoje pilno teritorijos išvalymo turi būti palaikomas pakankamas teritorijos švaros lygis (nurodomos sistemos modeliavusio tyrėjo). Keičiant tokius parametrus kaip užterštumas (*Pollution rate*), robotų gedimo tikimybė ir taisymo laikas (*Maintenance limit* ir *Maintenance time*), buvo dinamiškai tiriama robotų-darbininkų būsena – kiek robotų realiu laiku yra laukimo (*Idle*), darbo (*Working*) arba gedimo (*Maintained*) būsenoje; sektorių statusas – kiek sektorių yra išvalyti (*Clean*), valomi (*Dirty*) arba nešvarūs (*Processed*).

# 6 Susiję darbai ir pasiektų rezultatų analizė

## 6.1 Susiję darbai

Teritoriją valanti atspari gedimams (ang. *resilient*) daugiarobotė sistema buvo formaliai verifikuota tyrime, kurį atliko Åbo Akademi University mokslininkai [TPT+13]. Sistemą, kurios pagrindinis tikslas yra pilnai išvalyti jai priskirtą teritoriją, sudarė dviejų tipų robotai – valdymo stotys ir robotai-darbininkai. Sistema buvo formaliai modeliuojama naudojant Event-B bei verifikuota taikant tikimybinį modelių patikrinimą PRISM įrankiu siekiant išsiaiškinti, kokia įtaką pilnam sistemos darbo atlikimui per nurodytą laiką turi tokie parametrai kaip valdymo stočių ir robotų-darbininkų gedimo dažnis, robotų-darbininkų darbo (valymo) ir sistemos rekonfigūravimo sparta ir t.t.

Tyrime, atliktame baigiamojo bakalauro darbo metu [Dau21], panašios struktūros adaptyvi kompiuterinė sistema buvo formaliai verifikuojama taikant modelių patikrinimo metodą $TLA^+$ Toolbox aplinkoje. Buvo siekiama įrodyti, jog apibrėžta sistema pasižymi tokiomis būtinomis savybėmis, kaip klaidų toleravimas, dinaminė rekonfigūracija, duomenų integralumas ar kiekvienos valdymo stoties lokalaus vaizdo korektiškumas realios situacijos teritorijoje atžvilgiu. Lyginant su Åbo Akademi University mokslininkų atliktu tyrimu, analizuojama sistema buvo papildyta abiejų robotų tipų gedimo galimybe.

Šiame baigiamajame magistro darbe buvo formaliai modeliuojama ir verifikuojama patobulinta ir praplėsta (lyginant su minėtais tyrimais) adaptyvi kompiuterinė teritoriją valanti robotų sistema naudojant modelių patikrinimo įrankį UPPAAL. Sistemos modelis buvo papildytas naujais agentų tipais (robotu-mechaniku, robotu-skautu, sistemos komponenčių išjungimo mechanizmu ir t.t.) siekiant sumodeliuoti detalesnį ir tikslesnį sistemos atitikmenį. Naudojant statistinį modelių patikrinimą ir UPPAAL SMC plėtinį, buvo atlikti eksperimentai siekiant įvertinti tikimybę, jog nurodyta sistemos konfigūracija pilnai atliks darbą per nurodytą laiko tarpą, bus apibrėžtą laiką klaidos būsenoje arba suges viena ar daugiau sistemos komponenčių. Be to, buvo atliekamos simuliacijos siekiant įvertinti robotų-darbininkų energijos sunaudojimą ir valdymo stočių gedimo laikotarpį, bandoma išsiaiškinti, kiek maksimaliai ir minimaliai gali būti išvalyta sektorių per nurodytą laikotarpį.

## 6.2 Pasiektų rezultatų analizė

Tyrime buvo analizuojamos adaptyvios kompiuterinės sistemos, pasižyminčios hibridine

architektūra, modelis. Modelio struktūra apjungia dvi plačiai naudojamas sistemos organizavimo strategijas – hierarchinį bei saviorganizuojantį. Modeliuojamą sistemą sudaro skirtingų tipų komponentės, sudarančios kelis sistemos sluoksnius – tačiau tame pačiame sluoksnyje esančių agentų autonomiškumo lygis yra identiškas. Tyrimo rezultatai leidžia teigti, jog toks sistemos organizavimo būdas turi potencialo – sukurtas tokios architektūros pagrindu modelis yra stabilus, nepasižymi netikėtais darbo sutrikimais, veikia prognozuojamai bei efektyviai. Hibridinė architektūra pasižymi hierarchinės strategijos privalumais – pvz. moduliariškumu (ang. *modularity*) – pati sistema ir jai skirta užduotis yra išskaidoma į mažesnius, bendru atveju tarpusavy nesusiduriančius, modulius, kas leidžia vienos valdymo stoties gedimo atveju, kitoms valdymo stotims tęsti darbą įprastu režimu. Tuo pačiu, centralizuotos architektūros trūkumai – pvz. lankstumo (ang. *flexibility*) stoka gedimo atveju – yra niveliuojama saviorganizuojančios strategijos dėka – sugedus valdymo stočiai, jos robotai-darbininkai ir sektoriai yra optimaliai padalinami tarp kitų, veiksnių valdymo stočių. Tačiau hibridinė struktūra taip pat „paveldi“ abiejų jungiamų architektūrų trūkumus – tokia strategija sudarytos sistemos pvz. taip pat susiduria su, pvz., galimomis komponenčių komunikacijos problemomis.

Tikimybinio sistemos pilno darbo atlikimo (5.2.2.1 eksperimentas) ir visų valdymo stočių darbo pabaigimo (5.2.2.2) įvertinimo tyrimai buvo pratęsti, siekiant patyrinėti, kokią įtaką rezultatams turi parametrų kaita. Iš pradžių buvo siekiama apibrėžti eksponentinio rodiklio, nurodančio, kaip greitai esant galimybei yra paliekama dabartinė padėtis, įtaką tikimybei, jog per 500 laiko vienetų visa teritorija bus pilnai išvalyta. Analizuojant gautus rezultatus (žr. *Lentelė 1*), galima prieiti išvados, kad bendro, nurodomo daugumai padėčių rodiklio *normRate* (tyrimuose nurodyta reikšmė 1), klaidos padėčių rodiklio *errorRate* (0,5) ir robotų-darbininkų tausojimo režimo padėčių rodiklio *saveRate* (0,25) įtaka tikslo atlikimui per nurodytą laiko tarpą tikimybei yra minimali. Antra vertus, pokyčiai turi pastebimą įtaką laiko vienetų, reikalingų atlikti darbui, vidurkiui – kuo mažesni eksponentiniai rodikliai, tuo vėliau procesai pereina tarp būsenų, todėl ilgėja ir bendras sistemos darbo laiko vidurkis.

| *normRate* | *errorRate* | *saveRate* | *Tikimybė* | *Vidurkis* | *Verifikavimo laikas* |
|---|---|---|---|---|---|
| 0.5 | 0.25 | 0.125 | [0,989969; 0,994865] | 195 | 2 762,871 s |
| *1* | *0.5* | *0.25* | *[0,989159; 0,994279]* | *98* | *2 260,776 s* |
| 1 | 0.66 | 0.33 | [0,992079; 0,996351] | 85 | *2 516,535 s* |
| 1 | 1 | 1 | [0,99746; 0,999548] | 58 | 2 073,058 s |
| 4 | 2 | 1 | [0,994123; 0,997701] | 25 | 2 948,178 s |

*Lentelė 1. Tikimybinio pilno darbo atlikimo per 500 laiko vienetų priklausymo nuo eksponentinio rodiklio įvertinimo tyrimo rezultatų suvestinė*

Taip pat buvo siekiama įvertinti, kokią įtaką tiek sistemos užduoties atlikimui, tiek visų robotų darbo užbaigimo tikimybei turi stoties gedimo dažnį modeliuojančio parametro STATION_P (tyrimuose nurodyta reikšmė 200). Panašiai kaip ir eksponentinių rodiklių atveju, valdymo stoties gedimų pagausėjimas neturėjo įtakos pilno teritorijos išvalymo tikimybei (žr. *Lentelė 2*), kas gali būti laikoma dideliu modeliuojamos sistemos pliusu bei optimalaus resursų perskyrimo mechanizmo darbo rezultatu. Antra vertus, valdymo stočių gedimų padidėjimas turi pastebimą įtaką savalaikiam komponenčių darbo užbaigimui.

| ***STATION_P*** | ***Visų valdymo stočių darbo užbaigimo tikimybė*** | ***Pilno teritorijos išvalymo tikimybė*** | ***Verifikavimo laikas*** |
|---|---|---|---|
| 50 | [0,697907; 0,723456] | [0,994615; 0,998008] | 2 509,702 s |
| 100 | [0,832752; 0,853262] | [0,991019; 0,995613] | 2 362,525 s |
| *200* | *[0,915019; 0,930104]* | [0,989159; 0,994279] | *2 260,776 s* |
| 500 | [0,956319; 0,967158] | [0,989969; 0,994865] | 1 670,539 s |

*Lentelė 2. Tikimybinio pilno darbo atlikimo ir visų valdymo stočių darbo pabaigimo per 500 laiko vienetų priklausymo nuo valdymo stoties gedimo dažnį modeliuojančio parametro tyrimo rezultatų suvestinė*

Pradinės adaptyvios kompiuterinės sistemos modelio versijos analizės metu (verifikuojant taikant klasikinį modelių patikrinimo metodą) buvo pastebėtas potencialus valdymo stočių lokalaus vaizdo apsikeitimo scenarijus, kuriuo sekant, nepaisant visų sektorių išvalymo, nei viena valdymo stotis pagal savo lokalų vaizdą nelaikytų teritorijos pilnai išvalytos. Ši problema tapo aktuali dėl asmeninio komunikacijos tipo (kai viena valdymo stotis siunčia pranešimus kitai pasirinktai valdymo stočiai); ji tampa mažiau tikėtina naudojant komunikaciją transliavimo pagrindu (kai pranešėjas siunčia pranešimą visiems klausytojams), tačiau išlieka teoriškai įmanoma, kadangi potencialiai yra įmanoma, jog nei vienas iš adresatų pranešimo negaus. Eksperimentai patvirtino, jog efektyviu tokio pobūdžio problemos sprendimu komunikacijos dažnio nustatymas, atsižvelgiant į tai kaip seniai buvo paskutinį kartą komunikuojama. UPPAAL modeliavimo atveju tai buvo realizuota į valdymo stotį integruojant laikmatį, fiksuojantį laiką nuo paskutinio stoties pranešimo išsiuntimo. Teoriškai, pilnai sugedus komunikacijai tarp kompiuterinės sistemos komponenčių, galimybė valdymo stotims negauti naujausių atnaujinimų iš kitų išlieka, tačiau tokia tikimybė yra pastebimai sumažinama integravus laikmatį.

Statistinio modelių patikrinimo tyrimo metu buvo pastebėta viena iš potencialių modelio (ir visos sistemos) trūkumų – visoms kompiuterinės sistemos komponentėms nuolat gendant, apibrėžti laiką, reikiamą sistemai pilnai atlikti darbą, yra sudėtinga. Šiai problemai išspręsti galima dvejais būdais: teoriniu, abstrahuojantis nuo problemos ir pridedant prielaidą, jog komponentė gali sugesti tik bent minimaliai atlikusi darbą (pvz. valdymo stotis gali sugesti tik priskyrusi bent vieną sektorių robotui-darbininkui; robotas-darbininkas gali sugesti išvalęs bent vieną sektorių, po sutaisymo komponentė turi būti veiksni bent nurodytą laiko vienetų skaičių ir t.t.) arba praktiniu, įvedant naują agentą (pvz. robotą, atliekantį darbą tik sugedus visoms pagrindinėms komponentėms, ir „pavaduojantį“ valdymo stotį arba robotą-darbininką iki to laiko, kol bent vienas bus sutaisytas).

Modeliuojant kompiuterinę sistemą taip pat yra svarbu nustatyti procesų veikimo pradžios eiliškumą. Plečiant šiame tyrime analizuotą sistemos modelį, buvo nurodyta, jog valdymo stotys ir robotai-darbininkai gali pradėti darbą (pereiti iš pradinės būsenos) tik tada, kai visos sistemos reagavimą į gedimus užtikrinančios komponentės (robotas-mechanikas, robotas-skautas, resursų perskirstymo mechanizmas) jau pradėjo darbą. Tai implikuoja, jog ir procesas, modeliuojantis valdymo stočių ir robotų-mechanikų gedimus, turi prasidėti vėliau (t.y. procesas gali pereiti iš pradinės būsenos tik tada, kai jau darbo režime yra bent viena valdymo stotis). Tiek robotų, tiek gedimo proceso priklausymas nuo sistemos reagavimą į gedimus užtikrinančių komponenčių gali būti laikomas ne prielaida, o loginiu sąryšiu – eksploatuojant kompiuterinę sistemą realiame gyvenime prasminga įjungti gedimus apdorojančius robotus, ir tik po to faktišką darbą atliekančias komponentes, kurios, savo ruožtu, sugesti gali tik būdamos aktyvioje darbo fazėje. Valdymo sistemų teorijoje ir praktikoje tokie plačiai naudojami komponentai yra vadinami „šaltais atsarginiais“ (ang. *cold spares*).

Atliktas tyrimas taip pat iliustruoja verifikavimo įrankių (modelių patikrintojo UPPAAL ir plėtinio statistiniam modelių patikrinimui UPPAAL SMC) tinkamumą adaptyvių kompiuterinių sistemų formaliam modeliavimui ir verifikavimui. Klasikinis modelių patikrinimas leido verifikuoti esminių reikalavimų sistemai tenkinimą – tačiau svarbu pabrėžti, jog dėl įrankio apribojimų (būsenų sprogimo) modelis turėjo būti abstraktus bei pasižymėti tik turinčiomis tiesioginę įtaką reikalavimų patenkinimui savybėmis, o modelio konfigūracija (ypač agentų skaičius) būti minimali. Statistinio modelių patikrinimo metodo taikymas leido verifikuoti žymiai detalesnę, tiek konfigūracijos, tiek apibrėžtų komponenčių ir jų savybių, prasme sistemos modelio versiją. Taip pat buvo pastebėta, kad, siekiant tyrinėti sudėtingas sistemas, galinčias veikti neapibrėžtą laiko tarpą, tinkamu metodu gali būti simuliavimas, leidžiantis tyrimo metu keisti parametrų reikšmes bei tokiu būdu analizuoti jų įtaką tiriamojo objektui darbui.

Adaptyvios kompiuterinės sistemos tyrimas formaliai verifikuojant gali būti tęsiamas, detalizuojant sistemos modelį, mažinant abstrakcijos lygį. Vienas iš ypatingai svarbių aspektų, užtikrinančių sklandų sistemos darbą, yra komunikacija tarp sistemos komponenčių. Adaptyvios kompiuterinės sistemos analizę galima plėsti siekiant išsiaiškinti, kokie komunikavimo protokolai užtikrina tokių sistemų poaibiui didžiausią pranešimų perdavimo saugumą bei pasiekiamumą. Kitas svarbus sistemos patobulinimas galėtų būti susijęs su minimalaus darbo atlikimo užtikrinimu. Dabartinėje sistemos versijoje esti galimybė sistemos komponentėms gesti neribotą kiekį kartų – todėl teoriškai įmanoma, jog neapibrėžtą laiko tarpą sistema neišvalys nei vieno sektoriaus. Šią problemą galima išspręsti pvz. įvedant naujo roboto tipą, kuris veiktų tik kai visos valdymo stotys ar robotai-darbininkai būtų sugedęs. Kitas sprendimas būtų roboto-skauto bei roboto-mechaniko optimizavimas, tobulinant komunikaciją tarp komponenčių ir valdymo stočių.

## Išvados

Baigiamajame magistro darbe buvo atliktas tyrimas, kurio metu buvo sukurtas ir taikant statistinį modelių patikrinimo metodą verifikuotas formalus adaptyvios kompiuterinės sistemos modelis. Remiantis ankstesniuose darbo dalyse atlikta mokslinės literatūros analize buvo pasiūlyta hibridinė sistemos architektūra, apjungianti hierarchinį ir saviorganizuojantį modelius. Taip pat buvo apibūdinti pagrindiniai reikalavimai, keliami tokio tipo sistemoms, charakteristikos, kuriomis jo pasižymi bei verifikuotinos savybės, kurios buvo tiriamos. Pagrindinėmis sistemos charakteristikomis yra identifikuojami: distributyvumas, klaidų toleravimas bei dinaminė rekonfigūracija. Bendrai tokio tipo sistemoms keliami reikalavimai yra pilnas darbo atlikimas bei gebėjimas palaikyti sklandų darbą įvykus gedimams. Tiriamam sistemų poaibiui yra keliamas autonomiškumo reikalavimas – sistemos posistemės savarankiškai atlieka savo darbo dalį, be išorinės kontrolės.

Iškelti darbo uždaviniai buvo sėkmingai įvykdyti. Buvo atlikta mokslinės literatūros bei eksperimentų tiriant ir formaliai modeliuojant panašias arba artimas analizuojamai adaptyvias sistemas analizė. Rezultatai sudarė teorinį pagrindą darbo tyrimui – palyginus egzistuojančius metodus ir aplinkas, eksperimentams buvo pasirinktas modelio patikrinimo įrankis UPPAAL bei plėtinys UPPAAL SMC, skirtas statistiniam modelių patikrinimui. Esminiai ir papildomi pasirinktos kompiuterinės sistemos reikalavimai bei verifikuotinos savybės buvo išsiaiškintos ir aprašytos, o charakteristikos – identifikuotos ir įtrauktos į formalų tiriamos sistemos modelį.

Sėkmingai atlikto modelio verifikavimo pagrindu gauti tyrimo rezultatai buvo išanalizuoti, keičiant sistemos parametrus, tikslinant reikalavimus, apibrėžiant ir aprašant kartu su kitais darbo metu sukauptais pastebėjimais ir rekomendacijomis verifikuojamos sistemos kūrimui ir tolimesniems tyrimams. Tyrimą galima tęsti, koncentruojantis į kompiuterinės sistemos komponenčių komunikacijos efektyvumo bei minimalaus darbo per nurodytą laiko tarpą užtikrinimą. Tam reikėtų dar vieno sistemos modelio išplėtimo, galimo sistemos reikalavimų patikslinimo bei naujų prielaidų apie sistemos darbą suformulavimo.

## Šaltinių sąrašas


[ALN+05] J.R.Abrial, M.K.Lee, D.S.Neilson, P.N.Scharbach, I.H.Sørensen. The B-method. In: *VDM'91 Formal Software Development Methods: 4th International Symposium of VDM Europe Noordwijkerhout, The Netherlands, October 21–25, 1991 Proceedings*, Springer Berlin Heidelberg, 2005, pp. 398-405.

[Any22] The AnyLogic Company. AnyLogic, 2022. [Žiūrėta 2022-06-12] URL: <https://www.anylogic.com>

[BaK08] C.Baier, J.-P.Katoen. Principles of Model Checking. The MIT Press, Cambridge, Massachusetts, 2008, pp. 1-16. ISBN: 978-0-262-02649-9

[BDL04] G.Behrmann, A.David, K.G.Larsen. A Tutorial on Uppaal. Formal Methods for the Design of Real-Time Systems. In; *Lecture Notes in Computer Science*, vol 3185. Springer Berlin Heidelberg, 2004. DOI: 10.1007/978-3-540-30080-9_7

[BDL+12a] P.Bulychev, A.David, K.G.Larsen, A.Legay, M.Mikučionis, B.D.Poulsen. Checking and distributing statistical model checking. In: *NASA Formal Methods: 4th International Symposium*, Springer Berlin Heidelberg, 2012, pp. 449-463. DOI: 10.1007/978-3-642-28891-3_39

[BDL+12b] P.Bulychev, A.David, K.G.Larsen, M.Mikučionis, B.D.Poulsen, A.Legay, Z.Wang. UPPAAL-SMC: Statistical Model Checking for Priced Timed Automata. QAPL, 2012. DOI: 10.4204/EPTCS.85.1

[BDT99] E.Bonabeau, M.Dorigo, G.Theraulaz. Swarm Intelligence: From Natural to Artificial Systems. Oxford University Press, New York, USA. DOI: 10.1093/oso/9780195131581.001.0001

[Bje05] P.Bjesse. What is formal verification? *ACM SIGDA Newslett.*, 35(24), 2005. DOI: 10.1145/1113792.1113794

[BLL+96] J.Bengtsson, K.Larsen, F.Larsson, P.Pettersson, W.Yi. UPPAAL – a tool suite for automatic verification of real-time systems. In: *Hybrid Systems III*, Springer, Berlin, 1996, pp. 232-243. DOI: 10.1007/BFb0020949

[But16] R.Butler. What is Formal Methods? US Government, NASA. [Žiūrėta 2021-12-28] URL: <https://shemesh.larc.nasa.gov/fm/fm-what.html>

[CaV09] M.Casadei, M.Viroli. Using probabilistic model checking and simulation for designing self-organizing systems. In: *Proceedings of the 2009 ACM symposium on Applied Computing*, New York, NY, USA, 2009, pp. 2103–2104.

[CGK+18] E.M.Clarke Jr, O.Grumberg, D.Kroening, D.Peled, H.Veith. Model checking. MIT press, 2018. ISBN: 9780262349451

[Dau21] D.Daukševič. Kompiuterinių sistemų verifikavimas taikant modelių patikrinimo metodą $TLA^+$ Toolbox aplinkoje. Baigiamasis bakalauro darbas, Vilniaus Universitetas, Vilnius, 2021.

[Deb18] H.Debbi. Counterexamples in Model Checking-A survey. In: *Informatica*, *42*(2), 2018.

[DHS+03] K.Driscoll, B.Hall, H.Sivencrona, P.Zumsteg. Byzantine Fault Tolerance, from Theory to Reality. In: *Anderson, S., Felici, M., Littlewood, B. (eds) Computer Safety, Reliability, and Security*. SAFECOMP 2003. Lecture Notes in Computer Science, vol 2788. Springer, Berlin, Heidelberg, 2003. DOI: 10.1007/978-3-540-39878-3_19

[DLL+15] A.David, K.G.Larsen, A.Legay, M.Mikučionis, D.B.Poulsen. UPPAAL SMC tutorial. International Journal on Software Tools for Technology Transfer, 17(4), pp. 397–415, 2015. DOI: 10.1007/s10009-014-0361-y

[ECP18] ECPI University. What are Computer-Based information systems? [Žiūrėta 2022-01-09] URL: <https://www.ecpi.edu/blog/what-are-computer-based-information-systems>

[Gri21] I.Grigoryev. AnyLogic in three days. A quick course in simulation modeling. Fifth edition, 2021.

[Her90] D.W.Heermann. Computer-simulation methods, Springer Berlin Heidelberg, 1990, pp. 8-12. ISBN: 978-3-642-75448-7

[HiC10] M.Hinchey L.Coyle. Evolving Critical Systems: A Research Agenda for Computer-Based Systems. *17th IEEE International Conference and Workshops on Engineering of Computer Based Systems*, pp. 430-435. Oxford, United Kingdom, 2010. DOI: 10.1109/ECBS.2010.56.

[IfW12] U.M.Iftikhar, D.Weyns. A Case Study on Formal Verification of Self-Adaptive Behaviours in a Decentralized System. In Proceedings FOCLASA 2012. DOI: 10.4204/EPTCS.91.4

[KrK88] K.L.Kraemer, J.L.King. Computer-based systems for cooperative work and group decision making. In: *ACM Computing Surveys*, 20(2), pp. 115-146. 1988. DOI: 10.1145/46157.46158

[Lam02] L.Lamport. Specifying systems: the TLA+ language and tools for hardware and software engineers. Microsoft Research, 2002.

[LSP82] L.Lamport, R.Shostak, M.Pease. The Byzantine Generals Problem. In: *ACM Transactions on Programming Languages and Systems*, 4(3): 382– 401. 1982. DOI: 10.1145/357172.357176

[LTG+14] L.Laibinis, E.Troubitsyna, Z.Graja, F.Migeon, A.H.Kacem. Formal Modelling and Verification of Cooperative Ant Behaviour in Event-B. In: Giannakopoulou, D., Salaün, G. (eds) Software Engineering and Formal Methods. SEFM 2014. Lecture Notes in Computer Science, vol 8702. Springer, Cham.

DOI: 10.1007/978-3-319-10431-7_29

[Mer00] S.Merz. Model Checking: A Tutorial Overview. Summer School on Modeling and Verification of Parallel Processes, pp. 3-38, 2000. DOI: 10.5555/766794.766796

[NML+19] M.S.Nawaz, M.Malik, Y.Li, M.Sun, M.I.Ullah Lali. A Survey on Theorem Provers in Formal Methods. ArXiv, 2019. DOI: 10.48550/arXiv.1912.03028

[PMT20] A.Pappagallo, A.Massini, E.Tronci. Monte carlo based statistical model checking of cyber-physical systems: A review. In: *Information*, *11*(12), 588, 2020.

[PTL12] I.Pereverzeva, E.Troubitsyna, L.Laibinis. Development of Fault Tolerant MAS with Cooperative Error Recovery by Refinement in Event-B. In: *Proceedings of DS-Event-B 2012: Workshop on the experience of and advances in developing dependable systems in Event-B*, Kyoto, Japan, 2012. DOI: 10.48550/arXiv.1210.7035

[RaR20] S.Raju, K.Rytarowski. An introduction to Formal Verification for Software Systems. Moritz Systems. [Žiūrėta 2021-12-27]
URL: <https://www.moritz.systems/blog/an-introduction-to-formal-verification/ >

[Rus01] J.Rushby. Theorem Proving for Verification. In: Modeling and Verification of Parallel Processes. Springer, Berling, 2001, pp. 39-57.
DOI: 10.1007/3-540-45510- 8_2

[Sch01] J.M.Schumann. *Automated theorem proving in software engineering*. Springer Science & Business Media, 2001. ISBN: 3-540-67989-8

[Sys20] System Innovation. Formal Models. [Žiūrėta 2022-01-09]
URL: <https://www.systemsinnovation.io/post/formal-models>

[TPT+13] Tarasyuk, A., Pereverzeva, I., Troubitsyna, E., & Laibinis, L. Formal development and quantitative assessment of a resilient multi-robotic system. In: *Software Engineering for Resilient Systems: 5th International Workshop*, Springer Berlin Heidelberg, *2013*, *p*p. 109-124

[Upp19] UPPAAL Home Page. 2019. [Žiūrėta 2022-06-14]
URL: <https://uppaal.org>

[Upp21] UPPAAL Documentation. 2021. [Žiūrėta 2022-06-17]
URL: < https://docs.uppaal.org>

[Ver17] Verimag. Statistical Model Checking. 2017. [Žiūrėta 2022-06-04]
URL: <https://www-verimag.imag.fr/Statistical-Model-Checking-814.html>

[WLB+09] J.Woodcock, P.G.Larsen, J.Bicarregui, J.Fitzgerald. Formal methods: Practice and experience. ACM computing surveys (CSUR), 41(4), pp. 1-36. 2009.
DOI: 10.1145/1592434.1592436

## Priedas Nr. 1. Pradinė sistemos versija

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE nta PUBLIC '-//Uppaal Team//DTD Flat System 1.1//EN'
'http://www.it.uu.se/research/group/darts/uppaal/flat-1_2.dtd'>
<nta>
      <declaration>// Initial version of an Adaptive Computer-Based System

// Place global declarations here.
const int NUM_OF_STATIONS = 2;
const int NUM_OF_WORKERS  = 4;
const int NUM_OF_SECTORS  = 9;

const int MAX_DIRT = 1;

typedef int[0, NUM_OF_STATIONS-1] station_t;
typedef int[0, NUM_OF_WORKERS-1]  worker_t;
typedef int[0, NUM_OF_SECTORS-1]  sector_t;

typedef int[0, MAX_DIRT] dirt_t;

const int MAX_SILENCE = 20;

typedef struct
{
    bool terminated;
} controlStation_t;

typedef struct
{
    bool terminated;
} robotWorker_t;

controlStation_t station[NUM_OF_STATIONS] = { {false}, {false} };
robotWorker_t    worker[NUM_OF_WORKERS]   = { {false}, {false}, {false}, {false}
};

broadcast chan turnoff;

chan close[station_t],
     work[worker_t],
     done[worker_t],
     update[sector_t];

// Global variables
dirt_t sectorClean[sector_t] = { 0, 0, 0, 0, 0, 0, 0, 0, 0 };

// Station variables
station_t workerStation[worker_t] = { 0, 0, 1, 1 };
station_t sectorStation[sector_t] = { 0, 0, 0, 0, 1, 1, 1, 1, 0 };

bool localView[station_t][sector_t] = { { 0, 0, 0, 0, 0, 0, 0, 0, 0 },
                                        { 0, 0, 0, 0, 0, 0, 0, 0, 0 } };

bool sectorAssigned[sector_t] = { 0, 0, 0, 0, 0, 0, 0, 0, 0 };
bool workerAssigned[worker_t] = { 0, 0, 0, 0 };

// Worker variables
sector_t workerSector[worker_t] = { 0, 0, 0, 0 };

bool allStationsTerminated()
{
    return forall(s : station_t) station[s].terminated;
}

bool allWorkersTerminated()
```

```
{
    return forall(w : worker_t) worker[w].terminated;
}
</declaration>
      <template>
            <name x="5" y="5">ControlStation</name>
            <parameter>const station_t id</parameter>
            <declaration>// Place local declarations here.
bool othersInformed = true;

clock c;

bool checkAssignment(sector_t _sector, worker_t _worker)
{
    return (workerStation[_worker] == id &amp;&amp;
            sectorStation[_sector] == id &amp;&amp;
            !sectorClean[_sector]        &amp;&amp;
            !workerAssigned[_worker]     &amp;&amp;
            !sectorAssigned[_sector]);
}

void assignWork(sector_t _sector, worker_t _worker)
{
    workerSector[_worker]   = _sector;
    workerAssigned[_worker] = true;
    sectorAssigned[_sector] = true;
}

void acceptWork(worker_t _worker)
{
    localView[id][workerSector[_worker]] = true;
    workerAssigned[_worker]              = false;
    othersInformed                       = false;
}

void updateLocalView(station_t _station)
{
    for(s : sector_t)
    {
        localView[id][s] = localView[id][s] || localView[_station][s];
    }
}

bool checkLocalView()
{
    return forall(s : sector_t) localView[id][s];
}

bool noResponse()
{
    for(s : station_t)
    {
        if (s != id)
        {
            if (!station[s].terminated)
            {
                return false;
            }
        }
    }
    return true;
}

void inform()
```

```
{
    othersInformed = true;
    c = 0;
}</declaration>
            <location id="id0" x="195" y="68">
                  <name x="178" y="85">begin</name>
            </location>
            <location id="id1" x="391" y="68">
                  <name x="357" y="43">idle</name>
            </location>
            <location id="id2" x="654" y="136">
                  <name x="671" y="128">terminate</name>
                  <committed/>
            </location>
            <location id="id3" x="654" y="255">
                  <name x="671" y="246">end</name>
            </location>
            <init ref="id0"/>
            <transition>
                  <source ref="id2"/>
                  <target ref="id3"/>
                  <label kind="synchronisation" x="662" y="187">close[id]!
</label>
            </transition>
            <transition>
                  <source ref="id1"/>
                  <target ref="id2"/>
                  <label kind="guard" x="450" y="111">checkLocalView()</label>
                  <label kind="synchronisation" x="492" y="128">update[id]!
</label>
            </transition>
            <transition>
                  <source ref="id1"/>
                  <target ref="id2"/>
                  <label kind="guard" x="441" y="43">checkLocalView() &amp;amp;&amp;amp;
noResponse()</label>
                  <nail x="654" y="68"/>
            </transition>
            <transition>
                  <source ref="id1"/>
                  <target ref="id1"/>
                  <label kind="guard" x="144" y="145">!othersInformed  ||
c &gt; MAX_SILENCE</label>
                  <label kind="synchronisation" x="161" y="187">update[id]!
</label>
                  <label kind="assignment" x="170" y="204">inform()</label>
                  <nail x="272" y="162"/>
                  <nail x="246" y="128"/>
            </transition>
            <transition>
                  <source ref="id1"/>
                  <target ref="id1"/>
                  <label kind="select" x="509" y="-84">w : worker_t</label>
                  <label kind="guard" x="509" y="-67">workerStation[w] ==
id</label>
                  <label kind="synchronisation" x="509" y="-50">done[w]?</label>
                  <label kind="assignment" x="509" y="-33">acceptWork(w)</label>
                  <nail x="484" y="-42"/>
                  <nail x="510" y="-8"/>
            </transition>
            <transition>
                  <source ref="id1"/>
                  <target ref="id1"/>
                  <label kind="select" x="356" y="272">s : station_t</label>
```

```
                  <label kind="guard" x="365" y="289">s != id</label>
                  <label kind="synchronisation" x="357" y="306">update[s]?
</label>
                  <label kind="assignment" x="331"
y="323">updateLocalView(s)</label>
                  <nail x="348" y="272"/>
                  <nail x="425" y="272"/>
            </transition>
            <transition>
                  <source ref="id1"/>
                  <target ref="id1"/>
                  <label kind="select" x="323" y="-195">s : sector_t, w :
worker_t</label>
                  <label kind="guard" x="331" y="-178">checkAssignment(s,
w)</label>
                  <label kind="synchronisation" x="365" y="-161">work[w]!
</label>
                  <label kind="assignment" x="348" y="-144">assignWork(s,
w)</label>
                  <nail x="357" y="-119"/>
                  <nail x="433" y="-119"/>
            </transition>
            <transition>
                  <source ref="id0"/>
                  <target ref="id1"/>
            </transition>
      </template>
      <template>
            <name>RobotWorker</name>
            <parameter>const worker_t id</parameter>
            <declaration>void cleanSector()
{
    sectorClean[workerSector[id]] = 1;
}</declaration>
            <location id="id4" x="0" y="0">
                  <name x="-17" y="17">start</name>
            </location>
            <location id="id5" x="204" y="-102">
                  <name x="221" y="-110">execute</name>
            </location>
            <location id="id6" x="-238" y="0">
                  <name x="-247" y="17">end</name>
            </location>
            <init ref="id4"/>
            <transition>
                  <source ref="id4"/>
                  <target ref="id6"/>
                  <label kind="synchronisation" x="-161" y="-42">turnoff?
</label>
                  <label kind="assignment" x="-195" y="-
25">worker[id].terminated = true</label>
            </transition>
            <transition>
                  <source ref="id5"/>
                  <target ref="id4"/>
                  <label kind="synchronisation" x="212" y="-51">done[id]!
</label>
                  <label kind="assignment" x="212" y="-34">cleanSector()</label>
                  <nail x="204" y="0"/>
            </transition>
            <transition>
                  <source ref="id4"/>
                  <target ref="id5"/>
                  <label kind="synchronisation" x="59" y="-76">work[id]?</label>
```

```
            </transition>
      </template>
      <template>
            <name>RingMechanism</name>
            <location id="id7" x="-476" y="-17">
                  <name x="-493" y="0">start</name>
            </location>
            <location id="id8" x="-212" y="-17">
                  <name x="-229" y="0">listen</name>
            </location>
            <location id="id9" x="25" y="-17">
                  <name x="-9" y="0">turnOffWorkers</name>
            </location>
            <location id="id10" x="289" y="-17">
                  <name x="280" y="0">end</name>
            </location>
            <init ref="id7"/>
            <transition>
                  <source ref="id9"/>
                  <target ref="id9"/>
                  <label kind="guard" x="-42" y="-229">!
allWorkersTerminated()</label>
                  <label kind="synchronisation" x="0" y="-212">turnoff!</label>
                  <nail x="-59" y="-187"/>
                  <nail x="110" y="-187"/>
            </transition>
            <transition>
                  <source ref="id8"/>
                  <target ref="id8"/>
                  <label kind="select" x="-255" y="-246">s : station_t</label>
                  <label kind="synchronisation" x="-246" y="-229">close[s]?
</label>
                  <label kind="assignment" x="-297" y="-
212">station[s].terminated = true</label>
                  <nail x="-289" y="-187"/>
                  <nail x="-136" y="-187"/>
            </transition>
            <transition>
                  <source ref="id9"/>
                  <target ref="id10"/>
                  <label kind="guard" x="93" y="-
42">allWorkersTerminated()</label>
            </transition>
            <transition>
                  <source ref="id8"/>
                  <target ref="id9"/>
                  <label kind="guard" x="-170" y="-
42">allStationsTerminated()</label>
            </transition>
            <transition>
                  <source ref="id7"/>
                  <target ref="id8"/>
                  <label kind="select" x="-391" y="-76">s : station_t</label>
                  <label kind="synchronisation" x="-383" y="-60">close[s]?
</label>
                  <label kind="assignment" x="-434" y="-
43">station[s].terminated = true</label>
            </transition>
      </template>
      <system>// Place template instantiations here.

ring = RingMechanism();

// List one or more processes to be composed into a system.
```

```
system ControlStation,
       RobotWorker,
       ring;
    </system>
      <queries>
            <query>
                  <formula>A[] (deadlock imply (forall(s : station_t)
ControlStation(s).end))</formula>
                  <comment>5.2.1.1. Pilno darbo atlikimo patikrinimas
</comment>
            </query>
            <query>
                  <formula>A[] (forall(s : station_t) forall(t : sector_t)
(localView[s][t] imply sectorClean[t]))</formula>
                  <comment>5.2.1.2  Valdymo stoÄ¾iÅ³ lokaliÅ³ vaizdÅ³
korektiÅ¡kumo patikrinimas</comment>
            </query>
      </queries>
</nta>
```

**Priedas Nr. 2. Pilna sistemos versija**

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE nta PUBLIC '-//Uppaal Team//DTD Flat System 1.1//EN'
'http://www.it.uu.se/research/group/darts/uppaal/flat-1_2.dtd'>
<nta>
      <declaration>// Global declarations
const int NUM_OF_STATIONS  =  4;
const int NUM_OF_WORKERS   = 12;
const int TERRITORY_LEN    =  5;
const int NUM_OF_MECHANICS =  4;

const int MAX_DIRT                = 10;
const int WORKER_ENERGY           = 25;
const int STATION_REPAIRMENT_TIME = 20;
const int WORKER_REPAIRMENT_TIME  = 10;
const int TIME_SINCE_LAST_UPDATE  = 20;

const int STATION_P = 200;
const int WORKER_P  = 50;
const int ERROR_P   = 1;

const double normRate  = 1.0;
const double errorRate = 0.5;
const double saveRate  = 0.25;

const int RECONFIGURATION_TYPE = 2;

// Type definitions
typedef int[0, NUM_OF_STATIONS-1]             station_t;
typedef int[0, NUM_OF_WORKERS-1]              worker_t;
typedef int[0, TERRITORY_LEN-1]               sector_t;
typedef int[0, TERRITORY_LEN*TERRITORY_LEN-1] territory_t;
typedef int[0, NUM_OF_MECHANICS-1]            mechanic_t;

typedef int[0, MAX_DIRT-1] dirt_t;
typedef int[0, 3]          error_t;
typedef int[0, 1]          reconfig_t;

typedef struct
{
    sector_t posX;
    sector_t posY;
    bool     localView[territory_t];
    bool     updateShared;
    bool     started;
    bool     active;
    bool     terminated;
} controlStation_t;

controlStation_t station[NUM_OF_STATIONS] = { {0, 0, { 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, 0, 0, 1, 0},
                                              {0, 4, { 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, 0, 0, 1, 0},
                                              {4, 0, { 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, 0, 0, 1, 0},
                                              {4, 4, { 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 }, 0, 0, 1, 0} };

typedef struct
{
    sector_t  posX;
    sector_t  posY;
    station_t station;
    bool      assigned;
    bool      started;
```

```
    bool      active;
    int       energy;
    bool      refill;
    bool      terminated;
} robotWorker_t;

robotWorker_t worker[NUM_OF_WORKERS] = { {0, 1, 0, 0, 0, 1, WORKER_ENERGY, 0,
0},
                                        {1, 0, 0, 0, 0, 1, WORKER_ENERGY, 0,
0},
                                        {1, 1, 0, 0, 0, 1, WORKER_ENERGY, 0,
0},

                                        {0, 3, 1, 0, 0, 1, WORKER_ENERGY, 0,
0},
                                        {1, 3, 1, 0, 0, 1, WORKER_ENERGY, 0,
0},
                                        {1, 4, 1, 0, 0, 1, WORKER_ENERGY, 0,
0},

                                        {3, 0, 2, 0, 0, 1, WORKER_ENERGY, 0,
0},
                                        {3, 1, 2, 0, 0, 1, WORKER_ENERGY, 0,
0},
                                        {4, 1, 2, 0, 0, 1, WORKER_ENERGY, 0,
0},

                                        {3, 4, 3, 0, 0, 1, WORKER_ENERGY, 0,
0},
                                        {3, 3, 3, 0, 0, 1, WORKER_ENERGY, 0,
0},
                                        {4, 3, 3, 0, 0, 1, WORKER_ENERGY, 0, 0}
};

typedef struct
{
    sector_t  posX;
    sector_t  posY;
    bool      stationToRepair;
    station_t stationId;
    bool      workerToRepair;
    worker_t  workerId;
    bool      started;
    bool      terminated;
} robotMechanic_t;

robotMechanic_t mechanic[NUM_OF_MECHANICS] = { { (TERRITORY_LEN-1)/2,
(TERRITORY_LEN-1)/2, 0, 0, 0, 0, 0, 0 },
                                               { (TERRITORY_LEN-1)/2,
(TERRITORY_LEN-1)/2, 0, 0, 0, 0, 0, 0 },
                                               { (TERRITORY_LEN-1)/2,
(TERRITORY_LEN-1)/2, 0, 0, 0, 0, 0, 0 },
                                               { (TERRITORY_LEN-1)/2,
(TERRITORY_LEN-1)/2, 0, 0, 0, 0, 0, 0 } };

// Channel declarations
broadcast chan giveResources[station_t],
               giveSector[station_t],
               update[station_t],
               call[sector_t],
               close[sector_t],
               stopWorker,
               stopReassignment,
               stopMechanic,
```

```
                stopScout,
                stopDeactivation,
                work[worker_t],
                done[worker_t];

// Instance declarations
dirt_t sectorStatus[sector_t][sector_t] = { { 9, 7, 6, 8, 8 },
                                            { 8, 7, 9, 7, 9 },
                                            { 7, 8, 9, 9, 7 },
                                            { 9, 7, 6, 9, 8 },
                                            { 8, 8, 7, 7, 6 } };

bool sectorAssigned[territory_t] = { 0, 0, 0, 0, 0,
                                     0, 0, 0, 0, 0,
                                     0, 0, 0, 0, 0,
                                     0, 0, 0, 0, 0,
                                     0, 0, 0, 0, 0 };

station_t sectorStation[territory_t] = { 0, 0, 0, 1, 1,
                                         0, 0, 1, 1, 1,
                                         0, 2, 2, 1, 3,
                                         2, 2, 2, 3, 3,
                                         2, 2, 3, 3, 3 };

territory_t workerSector[worker_t] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

bool reassignmentStarted    = false;
bool reassignmentTerminated = false;
bool scoutStarted           = false;
bool scoutTerminated        = false;
bool deactivationTerminated = false;

double    queryValue;

// METHODS
// Territory/sector conversion
territory_t convert(sector_t _x, sector_t _y)
{
    return _x * TERRITORY_LEN + _y;
}

sector_t getX(territory_t _t)
{
    return (_t / TERRITORY_LEN);
}

sector_t getY(territory_t _t)
{
    return (_t % TERRITORY_LEN);
}

bool sectorIsCleaned(territory_t _t)
{
    return (sectorStatus[getX(_t)][getY(_t)] == 0);
}

int countPath(sector_t _fromX, sector_t _fromY,
              sector_t _toX,   sector_t _toY)
{
    return fint(
                sqrt(
                  pow((_fromX - _toX), 2) + pow((_fromY - _toY), 2)
                )
           );
```

```
}

int[0, TERRITORY_LEN*TERRITORY_LEN] countStationSectors(station_t _station)
{
    int[0, TERRITORY_LEN*TERRITORY_LEN] numOfSectors = 0;

    for(sector : territory_t)
    {
        if (sectorStation[sector] == _station &amp;&amp;
            !station[_station].localView[sector])
        {
            numOfSectors = numOfSectors + 1;
        }
    }

    return numOfSectors;
}

int[0, TERRITORY_LEN*TERRITORY_LEN] countSystemSectors()
{
    int[0, TERRITORY_LEN*TERRITORY_LEN] numOfSectors = 0;

    for(s : station_t)
    {
        for(sector : territory_t)
        {
            if (sectorStation[sector] == s &amp;&amp;
                !station[s].localView[sector])
            {
                numOfSectors = numOfSectors + 1;
            }
        }
    }

    return numOfSectors;
}

int[0, NUM_OF_WORKERS] countStationWorkers(station_t _station)
{
    int[0, NUM_OF_WORKERS] numOfWorkers = 0;

    for(w : worker_t)
    {
        if (worker[w].station == _station)
        {
            numOfWorkers = numOfWorkers + 1;
        }
    }

    return numOfWorkers;
}

bool stationHasSectors(station_t _station)
{
    for(sector : territory_t)
    {
        if (sectorStation[sector] == _station &amp;&amp;
            !station[_station].localView[sector])
        {
            return true;
        }
    }

    return false;
```

```
}

bool stationHasWorkers(station_t _station)
{
    for(w : worker_t)
    {
        if (worker[w].station == _station)
        {
            return true;
        }
    }

    return false;
}

bool stationHasResources(station_t _station)
{
    return (countStationSectors(_station) &gt; 0) &amp;&amp;
(countStationWorkers(_station) &gt; 0);
}

station_t selectStation(station_t _station)
{
    for(s : station_t)
    {
        if ((station[s].active &amp;&amp; !station[s].terminated) &amp;&amp;
            (s != _station))
        {
            return s;
        }
    }
    return _station;
}

bool systemStarted()
{
    return (exists(s : station_t) station[s].started);
}

int e()
{
    return sum(w : worker_t) worker[w].energy;
}

double r()
{
    if (exists(s : station_t) !station[s].active)
    {
        return queryValue;
    }

    return 0.0;
}

bool allMechanicsStarted()
{
    return forall(m : mechanic_t) mechanic[m].started;
}

bool systemCanStart()
{
    return allMechanicsStarted() &amp;&amp; reassignmentStarted &amp;&amp;
scoutStarted;
}</declaration>
```

```
      <template>
            <name x="5" y="5">ControlStation</name>
            <parameter>const station_t id</parameter>
            <declaration>// Place local declarations here.
//bool othersInformed = true;

clock sinceInformed;

bool checkAssignment(territory_t _sector, worker_t _worker)
{
    return (station[id].active              &amp;&amp;
            sectorStation[_sector]  == id   &amp;&amp;
            worker[_worker].station == id   &amp;&amp;
            worker[_worker].active          &amp;&amp;
            !station[id].localView[_sector] &amp;&amp;
            !worker[_worker].assigned       &amp;&amp;
            !sectorAssigned[_sector]);
}

void start()
{
    station[id].started = true;
}

bool checkWorker(worker_t _worker)
{
    return (station[id].active &amp;&amp; worker[_worker].station == id);
}

void assignWork(territory_t _sector, worker_t _worker)
{
    workerSector[_worker] = _sector;
}

void acceptWork(worker_t _worker)
{
    station[id].localView[workerSector[_worker]] = true;
    worker[_worker].assigned                     = false;
    station[id].updateShared                     = false;
}

void updateLocalView(station_t _station)
{
    for(s : territory_t)
    {
        station[id].localView[s] = station[id].localView[s] ||
station[_station].localView[s];
    }

    station[_station].updateShared = true;
}

bool checkLocalView()
{
    return forall(s : territory_t) station[id].localView[s];
}

bool noResponse()
{
    for(s : station_t)
    {
        if (s != id)
        {
            if (!station[s].terminated)
```

```
            {
                return false;
            }
        }
    }
    return true;
}

bool allSectorsCleaned()
{
    for(s : territory_t)
    {
        if (sectorStation[s] == id &amp;&amp; !station[id].localView[s])
        {
            return false;
        }
    }

    return true;
}
</declaration>
            <location id="id0" x="781" y="-314">
                  <name x="764" y="-297">begin</name>
                  <label kind="exponentialrate" x="798" y="-
340">normRate</label>
            </location>
            <location id="id1" x="1062" y="-314">
                  <name x="1096" y="-306">idle</name>
                  <label kind="exponentialrate" x="977" y="-
314">normRate</label>
            </location>
            <location id="id2" x="1376" y="-314">
                  <name x="1367" y="-305">error</name>
                  <label kind="exponentialrate" x="1367" y="-
348">errorRate</label>
            </location>
            <location id="id3" x="926" y="-195">
                  <name x="849" y="-212">terminate</name>
                  <label kind="exponentialrate" x="935" y="-
187">normRate</label>
            </location>
            <location id="id4" x="926" y="-93">
                  <name x="917" y="-76">end</name>
            </location>
            <init ref="id0"/>
            <transition>
                  <source ref="id1"/>
                  <target ref="id1"/>
                  <label kind="guard" x="1156" y="-238">station[id].active
&amp;&amp;
(!station[id].updateShared ||
  allSectorsCleaned() ||
  sinceInformed &gt; TIME_SINCE_LAST_UPDATE)</label>
                  <label kind="synchronisation" x="1190" y="-161">update[id]!
</label>
                  <label kind="assignment" x="1173" y="-144">sinceInformed =
0</label>
                  <nail x="1113" y="-204"/>
                  <nail x="1156" y="-237"/>
            </transition>
            <transition>
                  <source ref="id1"/>
                  <target ref="id1"/>
                  <label kind="select" x="1147" y="-467">w : worker_t</label>
```

```
                  <label kind="guard" x="1147" y="-450">checkWorker(w)</label>
                  <label kind="synchronisation" x="1147" y="-433">done[w]?
</label>
                  <label kind="assignment" x="1147" y="-
416">acceptWork(w)</label>
                  <nail x="1130" y="-407"/>
                  <nail x="1164" y="-373"/>
            </transition>
            <transition>
                  <source ref="id1"/>
                  <target ref="id1"/>
                  <label kind="guard" x="824" y="-467">station[id].active
&amp;&amp;
(!stationHasResources(id) ||
  allSectorsCleaned())</label>
                  <label kind="synchronisation" x="850" y="-
407">giveResources[id]!</label>
                  <nail x="960" y="-373"/>
                  <nail x="994" y="-407"/>
            </transition>
            <transition>
                  <source ref="id3"/>
                  <target ref="id4"/>
                  <label kind="synchronisation" x="867" y="-153">close[id]!
</label>
            </transition>
            <transition>
                  <source ref="id1"/>
                  <target ref="id3"/>
                  <label kind="guard" x="866" y="-289">station[id].active
&amp;&amp;
checkLocalView()</label>
                  <label kind="synchronisation" x="900" y="-255">update[id]!
</label>
            </transition>
            <transition>
                  <source ref="id1"/>
                  <target ref="id1"/>
                  <label kind="select" x="1028" y="-127">s : station_t</label>
                  <label kind="guard" x="994" y="-110">station[id].active
&amp;&amp; s != id</label>
                  <label kind="synchronisation" x="1037" y="-93">update[s]?
</label>
                  <label kind="assignment" x="1011" y="-
76">updateLocalView(s)</label>
                  <nail x="1020" y="-127"/>
                  <nail x="1105" y="-127"/>
            </transition>
            <transition>
                  <source ref="id1"/>
                  <target ref="id1"/>
                  <label kind="select" x="986" y="-586">t : territory_t, w :
worker_t</label>
                  <label kind="guard" x="986" y="-569">checkAssignment(t,
w)</label>
                  <label kind="synchronisation" x="1037" y="-552">work[w]!
</label>
                  <label kind="assignment" x="1011" y="-535">assignWork(t,
w)</label>
                  <nail x="1020" y="-509"/>
                  <nail x="1096" y="-509"/>
            </transition>
            <transition>
                  <source ref="id2"/>
```

```
                  <target ref="id1"/>
                  <label kind="guard" x="1240" y="-
374">station[id].active</label>
                  <label kind="synchronisation" x="1240" y="-
357">giveResources[id]!</label>
                  <nail x="1198" y="-340"/>
            </transition>
            <transition>
                  <source ref="id1"/>
                  <target ref="id2"/>
                  <label kind="guard" x="1240" y="-297">!
station[id].active</label>
                  <nail x="1198" y="-289"/>
            </transition>
            <transition>
                  <source ref="id0"/>
                  <target ref="id1"/>
                  <label kind="guard" x="866" y="-356">systemCanStart()</label>
                  <label kind="assignment" x="900" y="-339">start()</label>
            </transition>
      </template>
      <template>
            <name>ReassignmentMechanism</name>
            <declaration>station_t currStation;

// ManagementProcesss

void initProcess()
{
    reassignmentStarted = true;
    queryValue          = random(10.0);
}

bool checkForStoppedStation(station_t _station)
{
    if (!station[_station].active ||
         station[_station].terminated)
    {
        return stationHasResources(_station);
    }
    return false;
}

void reassignWorkers(station_t _stationFrom, station_t _stationTo)
{
    for(w : worker_t)
    {
        if (worker[w].station == _stationFrom)
        {
            worker[w].station = _stationTo;
        }
    }
}

void reassignNearestWorker(station_t _stationFrom, station_t _stationTo)
{
    worker_t     nearestWorker;

    int[0, TERRITORY_LEN*TERRITORY_LEN]  nearestPath,
                                         newPath;

    bool         firstFound = false;

    for(w : worker_t)
```

```
    {
        if (worker[w].station == _stationFrom)
        {
            if (!firstFound)
            {
                nearestWorker = w;
                nearestPath = countPath(station[_stationTo].posX,
station[_stationTo].posY,
                                        worker[w].posX,
worker[w].posY);
                firstFound  = true;
            }
            else
            {
                newPath = countPath(station[_stationTo].posX,
station[_stationTo].posY,
                                    worker[w].posX,           worker[w].posY);

                if (newPath &lt; nearestPath)
                {
                    nearestWorker = w;
                    nearestPath   = newPath;
                }
            }
        }
    }

    worker[nearestWorker].station = _stationTo;
}

void reassignNearestSector(station_t _stationFrom, station_t _stationTo)
{
    territory_t    nearestSector;
    sector_t       sectorX,
                   sectorY;

    int[0, TERRITORY_LEN*TERRITORY_LEN]      nearestPath,
                   newPath;

    bool         firstFound = false;

    for(s : territory_t)
    {
        if (sectorStation[s] == _stationFrom)
        {
            if (!firstFound)
            {
                sectorX       = getX(s);
                sectorY       = getY(s);
                nearestSector = s;
                nearestPath = countPath(station[_stationTo].posX,
station[_stationTo].posY,
                                        sectorX,                  sectorY);
                firstFound  = true;
            }
            else
            {
                sectorX = getX(s);
                sectorY = getY(s);
                newPath = countPath(station[_stationTo].posX,
station[_stationTo].posY,
                                    sectorX,                  sectorY);

                if (newPath &lt; nearestPath)
```

```
                {
                    nearestSector = s;
                    nearestPath   = newPath;
                }
            }
        }
    }

    sectorStation[nearestSector] = _stationTo;
}

void reassignSectors(station_t _stationFrom, station_t _stationTo)
{
    for(sector : territory_t)
    {
        if (sectorStation[sector] == _stationFrom &amp;amp;&amp;amp;
            !station[_stationFrom].localView[sector])
        {
            sectorStation[sector] = _stationTo;
        }
    }
}

void reassignFromSelected()
{
    station_t stationTo = selectStation(currStation);

    reassignWorkers(currStation, stationTo);
    reassignSectors(currStation, stationTo);
}

void reassignFromEqually()
{
    station_t lastStation;
    int       numOfStations;

    if (exists(s : station_t) station[s].active)
    {
        numOfStations = (sum(s : station_t) station[s].active);

        for(stationTo : station_t)
        {
            if (station[stationTo].active &amp;amp;&amp;amp; (stationTo != currStation))
            {
                lastStation = stationTo;

                if (countStationWorkers(stationTo) &lt; NUM_OF_WORKERS /
numOfStations)
                {
                    reassignNearestWorker(currStation, stationTo);
                }
                if (countStationSectors(stationTo) &lt; (countSystemSectors() /
numOfStations))
                {
                   reassignNearestSector(currStation, stationTo);
                }
            }
        }
        reassignWorkers(currStation, lastStation);
        reassignSectors(currStation, lastStation);
    }
}

void reassignFrom()
```

```
{
    if (RECONFIGURATION_TYPE == 1)
    {
        reassignFromSelected();
    }
    else if (RECONFIGURATION_TYPE == 2)
    {
        reassignFromEqually();
    }
}

void reassignTo()
{
    for(stationFrom : station_t)
    {
        if (stationFrom != currStation)
        {
            if (checkForStoppedStation(stationFrom))
            {
                reassignWorkers(stationFrom, currStation);
                reassignSectors(stationFrom, currStation);
            }
            else
            {
                if (countStationWorkers(stationFrom) &gt; 1)
                {
                    reassignNearestWorker(stationFrom, currStation);
                }
                if (countStationSectors(stationFrom) &gt; 1)
                {
                    reassignNearestSector(stationFrom, currStation);
                }
            }
        }
    }
}

</declaration>
            <location id="id5" x="-2983" y="-858">
                  <name x="-3000" y="-850">begin</name>
                  <label kind="exponentialrate" x="-2991" y="-
892">normRate</label>
            </location>
            <location id="id6" x="-2567" y="-654">
                  <name x="-2584" y="-646">reasssignFromStopped</name>
                  <committed/>
            </location>
            <location id="id7" x="-2813" y="-858">
                  <name x="-2847" y="-850">idle</name>
                  <label kind="exponentialrate" x="-2779" y="-
850">normRate</label>
            </location>
            <location id="id8" x="-2567" y="-1054">
                  <name x="-2584" y="-1088">reassignToNeeding</name>
                  <committed/>
            </location>
            <location id="id9" x="-2550" y="-858">
                  <name x="-2533" y="-867">end</name>
            </location>
            <init ref="id5"/>
            <transition>
                  <source ref="id7"/>
                  <target ref="id9"/>
                  <label kind="synchronisation" x="-2737" y="-
```

```
901">stopReassignment?</label>
                  <label kind="assignment" x="-2771" y="-
884">reassignmentTerminated = true</label>
            </transition>
            <transition>
                  <source ref="id8"/>
                  <target ref="id7"/>
                  <label kind="assignment" x="-2745" y="-
1079">reassignTo()</label>
                  <nail x="-2813" y="-1055"/>
            </transition>
            <transition>
                  <source ref="id7"/>
                  <target ref="id8"/>
                  <label kind="select" x="-2643" y="-994">s : station_t</label>
                  <label kind="synchronisation" x="-2643" y="-
977">giveResources[s]?</label>
                  <label kind="assignment" x="-2643" y="-960">currStation =
s</label>
            </transition>
            <transition>
                  <source ref="id6"/>
                  <target ref="id7"/>
                  <label kind="assignment" x="-2754" y="-
646">reassignFrom()</label>
                  <nail x="-2813" y="-654"/>
            </transition>
            <transition>
                  <source ref="id5"/>
                  <target ref="id7"/>
                  <label kind="assignment" x="-2949" y="-
850">initProcess()</label>
            </transition>
            <transition>
                  <source ref="id7"/>
                  <target ref="id6"/>
                  <label kind="select" x="-2643" y="-773">s : station_t</label>
                  <label kind="guard" x="-2643" y="-
756">checkForStoppedStation(s)</label>
                  <label kind="assignment" x="-2643" y="-739">currStation =
s</label>
            </transition>
      </template>
      <template>
            <name>DeactivationProcess</name>
            <declaration>station_t currStation;
worker_t  currWorker;

bool pause()
{
    return ((forall(s : station_t) !station[s].active) &amp;&amp; (forall(w :
worker_t)  !worker[w].active));
}

bool checkStation(station_t _s)
{
    return (station[_s].started &amp;&amp;
            station[_s].active  &amp;&amp;
            !station[_s].terminated);
}

bool checkWorker(worker_t _w)
{
    return (worker[_w].started &amp;&amp;
```

```
            worker[_w].active  &amp;&amp;
            !worker[_w].refill  &amp;&amp;
            !worker[_w].terminated);
}</declaration>
            <location id="id10" x="-1419" y="-798">
                  <name x="-1436" y="-781">begin</name>
                  <label kind="exponentialrate" x="-1436" y="-
832">normRate</label>
            </location>
            <location id="id11" x="-1156" y="-756">
                  <name x="-1207" y="-765">idle</name>
                  <label kind="exponentialrate" x="-1131" y="-
765">normRate</label>
            </location>
            <location id="id12" x="-1420" y="-663">
                  <name x="-1429" y="-646">end</name>
            </location>
            <branchpoint id="id13" x="-1156" y="-1045">
            </branchpoint>
            <branchpoint id="id14" x="-1156" y="-467">
            </branchpoint>
            <init ref="id10"/>
            <transition>
                  <source ref="id11"/>
                  <target ref="id11"/>
                  <label kind="guard" x="-1037" y="-764">pause()</label>
                  <nail x="-1046" y="-807"/>
                  <nail x="-1046" y="-705"/>
            </transition>
            <transition>
                  <source ref="id11"/>
                  <target ref="id12"/>
                  <label kind="synchronisation" x="-1419" y="-
756">stopDeactivation?</label>
                  <label kind="assignment" x="-1445" y="-
739">deactivationTerminated = true</label>
            </transition>
            <transition>
                  <source ref="id10"/>
                  <target ref="id11"/>
                  <label kind="guard" x="-1360" y="-832">systemCanStart()
&amp;&amp;
systemStarted()</label>
            </transition>
            <transition>
                  <source ref="id13"/>
                  <target ref="id11"/>
                  <label kind="probability" x="-1003" y="-909">STATION_P</label>
                  <nail x="-1012" y="-901"/>
            </transition>
            <transition>
                  <source ref="id13"/>
                  <target ref="id11"/>
                  <label kind="assignment" x="-1428" y="-
994">station[currStation].active = false</label>
                  <label kind="probability" x="-1369" y="-909">ERROR_P</label>
                  <nail x="-1301" y="-901"/>
            </transition>
            <transition>
                  <source ref="id11"/>
                  <target ref="id13"/>
                  <label kind="select" x="-1148" y="-935">s : station_t</label>
                  <label kind="guard" x="-1148" y="-918">checkStation(s)</label>
                  <label kind="assignment" x="-1148" y="-901">currStation :=
```

```
s</label>
            </transition>
            <transition>
                  <source ref="id14"/>
                  <target ref="id11"/>
                  <label kind="probability" x="-1386" y="-611">WORKER_P</label>
                  <nail x="-1301" y="-603"/>
            </transition>
            <transition>
                  <source ref="id14"/>
                  <target ref="id11"/>
                  <label kind="assignment" x="-1080" y="-
535">worker[currWorker].active = false</label>
                  <label kind="probability" x="-1003" y="-620">ERROR_P</label>
                  <nail x="-1012" y="-611"/>
            </transition>
            <transition>
                  <source ref="id11"/>
                  <target ref="id14"/>
                  <label kind="select" x="-1241" y="-628">w : worker_t</label>
                  <label kind="guard" x="-1266" y="-611">checkWorker(w)</label>
                  <label kind="assignment" x="-1267" y="-594">currWorker :=
w</label>
            </transition>
      </template>
      <template>
            <name>RobotScout</name>
            <declaration>station_t     selectedStation;
worker_t      selectedWorker;
mechanic_t    selectedMechanic;

bool checkStationNotAssigned(station_t _s)
{
    return !(exists(m : mechanic_t) mechanic[m].stationToRepair &amp;&amp;
mechanic[m].stationId == _s);
}

bool checkScoutedStation(station_t _s)
{
    if (station[_s].started &amp;&amp;
        !station[_s].active  &amp;&amp;
        !station[_s].terminated)
    {
        return checkStationNotAssigned(_s);
    }
    else
    {
        return false;
    }
}

bool checkWorkerNotAssigned(worker_t _w)
{
    return !(exists(m : mechanic_t) mechanic[m].workerToRepair &amp;&amp;
mechanic[m].workerId == _w);
}

bool checkScoutedWorker(worker_t _w)
{
    if (worker[_w].started &amp;&amp;
        !worker[_w].active &amp;&amp;
        !worker[_w].refill &amp;&amp;
        !worker[_w].terminated)
    {
```

```
        return checkWorkerNotAssigned(_w);
    }
    else
    {
        return false;
    }
}

bool existsFreeMechanic()
{
    return (exists(m : mechanic_t) !mechanic[m].stationToRepair &amp;amp;&amp;amp; !
mechanic[m].workerToRepair);
}

void selectMechanic()
{
    for(m : mechanic_t)
    {
        if (!mechanic[m].stationToRepair &amp;amp;&amp;amp; !mechanic[m].workerToRepair)
        {
            selectedMechanic = m;
        }
    }
}

void assignStationToMechanic()
{
    selectMechanic();

    mechanic[selectedMechanic].stationToRepair = true;
    mechanic[selectedMechanic].stationId       = selectedStation;
}

void assignWorkerToMechanic()
{
    selectMechanic();

    mechanic[selectedMechanic].workerToRepair = true;
    mechanic[selectedMechanic].workerId       = selectedWorker;
}</declaration>
            <location id="id15" x="-977" y="-348">
                  <name x="-994" y="-331">begin</name>
                  <label kind="exponentialrate" x="-960" y="-
374">normRate</label>
            </location>
            <location id="id16" x="-603" y="-348">
                  <name x="-654" y="-348">idle</name>
                  <label kind="exponentialrate" x="-578" y="-
374">normRate</label>
            </location>
            <location id="id17" x="-603" y="-637">
                  <name x="-637" y="-672">checkStation</name>
                  <label kind="exponentialrate" x="-586" y="-
628">normRate</label>
            </location>
            <location id="id18" x="-892" y="-637">
                  <name x="-951" y="-671">reportStation</name>
                  <committed/>
            </location>
            <location id="id19" x="-603" y="-42">
                  <name x="-645" y="-25">checkWorker</name>
                  <label kind="exponentialrate" x="-586" y="-
68">normRate</label>
            </location>
```

```
            <location id="id20" x="-900" y="-42">
                  <name x="-968" y="-25">reportWorker</name>
                  <committed/>
            </location>
            <location id="id21" x="-246" y="-348">
                  <name x="-255" y="-331">end</name>
            </location>
            <init ref="id15"/>
            <transition>
                  <source ref="id16"/>
                  <target ref="id21"/>
                  <label kind="synchronisation" x="-450" y="-391">stopScout?
</label>
                  <label kind="assignment" x="-484" y="-374">scoutTerminated =
true</label>
            </transition>
            <transition>
                  <source ref="id20"/>
                  <target ref="id16"/>
                  <label kind="assignment" x="-952" y="-
195">assignWorkerToMechanic()</label>
            </transition>
            <transition>
                  <source ref="id19"/>
                  <target ref="id20"/>
                  <label kind="guard" x="-833" y="-
34">existsFreeMechanic()</label>
            </transition>
            <transition>
                  <source ref="id19"/>
                  <target ref="id16"/>
                  <label kind="guard" x="-510" y="-34">!
existsFreeMechanic()</label>
                  <nail x="-305" y="-42"/>
            </transition>
            <transition>
                  <source ref="id16"/>
                  <target ref="id19"/>
                  <label kind="select" x="-595" y="-195">w : worker_t</label>
                  <label kind="guard" x="-595" y="-
178">checkScoutedWorker(w)</label>
                  <label kind="assignment" x="-595" y="-161">selectedWorker =
w</label>
            </transition>
            <transition>
                  <source ref="id17"/>
                  <target ref="id16"/>
                  <label kind="guard" x="-493" y="-663">!
existsFreeMechanic()</label>
                  <nail x="-305" y="-637"/>
            </transition>
            <transition>
                  <source ref="id18"/>
                  <target ref="id16"/>
                  <label kind="assignment" x="-952" y="-
510">assignStationToMechanic()</label>
                  <nail x="-875" y="-620"/>
            </transition>
            <transition>
                  <source ref="id17"/>
                  <target ref="id18"/>
                  <label kind="guard" x="-816" y="-
663">existsFreeMechanic()</label>
            </transition>
```

```
            <transition>
                  <source ref="id16"/>
                  <target ref="id17"/>
                  <label kind="select" x="-595" y="-561">s : station_t</label>
                  <label kind="guard" x="-595" y="-
544">checkScoutedStation(s)</label>
                  <label kind="assignment" x="-594" y="-526">selectedStation =
s</label>
            </transition>
            <transition>
                  <source ref="id15"/>
                  <target ref="id16"/>
                  <label kind="assignment" x="-867" y="-340">scoutStarted =
true</label>
            </transition>
      </template>
      <template>
            <name>RobotMechanic</name>
            <parameter>const mechanic_t id</parameter>
            <declaration>sector_t    brokenStationX,
            brokenStationY;
territory_t pathToStation;

sector_t    brokenWorkerX,
            brokenWorkerY;
territory_t pathToWorker;

int         travelTime,
            workTime;

void setBrokenStation()
{
    brokenStationX = station[mechanic[id].stationId].posX;
    brokenStationY = station[mechanic[id].stationId].posY;
    pathToStation  =  fint(
                            sqrt(
                                pow((mechanic[id].posX -
station[mechanic[id].stationId].posX), 2) + pow((mechanic[id].posY -
station[mechanic[id].stationId].posY), 2)
                            )
                        );
    travelTime     = 0;
}

void setBrokenWorker()
{
    brokenWorkerX = worker[mechanic[id].workerId].posX;
    brokenWorkerY = worker[mechanic[id].workerId].posY;
    pathToWorker  = fint(
                         sqrt(
                             pow((mechanic[id].posX -
worker[mechanic[id].workerId].posX), 2) + pow((mechanic[id].posY -
worker[mechanic[id].workerId].posY), 2)
                         )
                      );
    travelTime    = 0;
}

void updatePosition(sector_t _x, sector_t _y)
{
    if (mechanic[id].posX &gt; _x)
    {
        mechanic[id].posX = mechanic[id].posX - 1;
    }
```

```
    else if (mechanic[id].posX &lt; _x)
    {
        mechanic[id].posX = mechanic[id].posX + 1;
    }
    else
    {
        if (mechanic[id].posY &gt; _y)
        {
            mechanic[id].posY = mechanic[id].posY - 1;
        }
        else if (mechanic[id].posY &lt; _y)
        {
            mechanic[id].posY = mechanic[id].posY + 1;
        }
    }

    travelTime = travelTime + 1;
}

void arrive(sector_t _x, sector_t _y)
{
    mechanic[id].posX = _x;
    mechanic[id].posY = _y;

    workTime = 0;
}

void repair()
{
    workTime = workTime + 1;
}

void resetInfo()
{
    travelTime     = 0;
    workTime       = 0;

    brokenStationX = 0;
    brokenStationY = 0;
    pathToStation  = 0;

    brokenWorkerX = 0;
    brokenWorkerY = 0;
    pathToWorker  = 0;

    mechanic[id].stationToRepair = false;
    mechanic[id].stationId       = 0;
    mechanic[id].workerToRepair  = false;
    mechanic[id].workerId        = 0;
}

void reactivateStation()
{
    station[mechanic[id].stationId].active = true;

    resetInfo();
}

void reactivateWorker()
{
    worker[mechanic[id].workerId].active = true;

    resetInfo();
}      </declaration>
```

```
            <location id="id22" x="-21794" y="-20680">
                  <name x="-21820" y="-20663">idle</name>
                  <label kind="exponentialrate" x="-21862" y="-
20706">normRate</label>
            </location>
            <location id="id23" x="-21454" y="-20799">
                  <name x="-21463" y="-20782">goToStation</name>
                  <label kind="exponentialrate" x="-21437" y="-
20825">normRate</label>
            </location>
            <location id="id24" x="-21454" y="-20562">
                  <name x="-21464" y="-20596">goToWorker</name>
                  <label kind="exponentialrate" x="-21531" y="-
20561">normRate</label>
            </location>
            <location id="id25" x="-20987" y="-20562">
                  <name x="-21021" y="-20596">repairWorker</name>
                  <label kind="exponentialrate" x="-20961" y="-
20553">normRate</label>
            </location>
            <location id="id26" x="-20987" y="-20799">
                  <name x="-21021" y="-20782">repairStation</name>
                  <label kind="exponentialrate" x="-20970" y="-
20825">normRate</label>
            </location>
            <location id="id27" x="-21454" y="-20680">
                  <name x="-21462" y="-20663">end</name>
            </location>
            <location id="id28" x="-22007" y="-20680">
                  <name x="-22024" y="-20663">begin</name>
                  <label kind="exponentialrate" x="-21990" y="-
20706">normRate</label>
            </location>
            <init ref="id28"/>
            <transition>
                  <source ref="id28"/>
                  <target ref="id22"/>
                  <label kind="assignment" x="-21989" y="-
20680">mechanic[id].started = true</label>
            </transition>
            <transition>
                  <source ref="id22"/>
                  <target ref="id27"/>
                  <label kind="synchronisation" x="-21633" y="-
20723">stopMechanic?</label>
                  <label kind="assignment" x="-21692" y="-
20706">mechanic[id].terminated = true</label>
            </transition>
            <transition>
                  <source ref="id26"/>
                  <target ref="id22"/>
                  <label kind="guard" x="-20842" y="-20910">workTime &gt;=
STATION_REPAIRMENT_TIME</label>
                  <label kind="assignment" x="-20842" y="-
20893">reactivateStation()</label>
                  <nail x="-20851" y="-20799"/>
                  <nail x="-20851" y="-20995"/>
                  <nail x="-21658" y="-20995"/>
            </transition>
            <transition>
                  <source ref="id26"/>
                  <target ref="id26"/>
                  <label kind="guard" x="-21122" y="-20986">workTime &lt;
STATION_REPAIRMENT_TIME</label>
```

```
                  <label kind="assignment" x="-21012" y="-
20969">repair()</label>
                  <nail x="-21055" y="-20944"/>
                  <nail x="-20928" y="-20944"/>
            </transition>
            <transition>
                  <source ref="id23"/>
                  <target ref="id23"/>
                  <label kind="guard" x="-21522" y="-20986">travelTime &lt;
pathToStation</label>
                  <label kind="assignment" x="-21624" y="-
20969">updatePosition(brokenStationX, brokenStationY)</label>
                  <nail x="-21522" y="-20944"/>
                  <nail x="-21395" y="-20944"/>
            </transition>
            <transition>
                  <source ref="id23"/>
                  <target ref="id26"/>
                  <label kind="guard" x="-21327" y="-20876">travelTime &gt;=
pathToStation</label>
                  <label kind="assignment" x="-21327" y="-
20850">arrive(brokenStationX, brokenStationY)</label>
            </transition>
            <transition>
                  <source ref="id25"/>
                  <target ref="id22"/>
                  <label kind="guard" x="-20842" y="-20477">workTime &gt;=
WORKER_REPAIRMENT_TIME</label>
                  <label kind="assignment" x="-20842" y="-
20460">reactivateWorker()</label>
                  <nail x="-20851" y="-20562"/>
                  <nail x="-20851" y="-20375"/>
                  <nail x="-21667" y="-20375"/>
            </transition>
            <transition>
                  <source ref="id25"/>
                  <target ref="id25"/>
                  <label kind="guard" x="-21131" y="-20400">workTime &lt;
WORKER_REPAIRMENT_TIME</label>
                  <label kind="assignment" x="-21012" y="-
20417">repair()</label>
                  <nail x="-21047" y="-20418"/>
                  <nail x="-20919" y="-20418"/>
            </transition>
            <transition>
                  <source ref="id24"/>
                  <target ref="id24"/>
                  <label kind="guard" x="-21514" y="-20417">travelTime &lt;
pathToWorker</label>
                  <label kind="assignment" x="-21624" y="-
20400">updatePosition(brokenWorkerX, brokenWorkerY)</label>
                  <nail x="-21522" y="-20418"/>
                  <nail x="-21386" y="-20418"/>
            </transition>
            <transition>
                  <source ref="id24"/>
                  <target ref="id25"/>
                  <label kind="guard" x="-21344" y="-20544">travelTime &gt;=
pathToWorker</label>
                  <label kind="assignment" x="-21344" y="-
20519">arrive(brokenWorkerX, brokenWorkerY)</label>
            </transition>
            <transition>
                  <source ref="id22"/>
```

```
                  <target ref="id24"/>
                  <label kind="guard" x="-21726" y="-
20587">mechanic[id].workerToRepair</label>
                  <label kind="assignment" x="-21692" y="-
20570">setBrokenWorker()</label>
            </transition>
            <transition>
                  <source ref="id22"/>
                  <target ref="id23"/>
                  <label kind="guard" x="-21692" y="-
20799">mechanic[id].stationToRepair</label>
                  <label kind="assignment" x="-21675" y="-
20782">setBrokenStation()</label>
            </transition>
      </template>
      <template>
            <name>TerminationMechanism</name>
            <declaration>bool stationCalled[station_t];

void initSession()
{
    for(s : station_t)
    {
        stationCalled[s] = false;
    }
}

void updateCall(station_t s)
{
    station[s].terminated = true;
    stationCalled[s]      = true;
}

bool allCalled()
{
    return forall(s : station_t) stationCalled[s];
}

bool allStationsTerminated()
{
    return forall(s : station_t) station[s].terminated;
}

bool allWorkersTerminated()
{
    return forall(s : worker_t) worker[s].terminated;
}

bool allMechanicsTerminated()
{
    return forall(m : mechanic_t) mechanic[m].terminated;
}</declaration>
            <location id="id29" x="-1326" y="-450">
                  <name x="-1343" y="-433">begin</name>
                  <label kind="exponentialrate" x="-1352" y="-
484">normRate</label>
            </location>
            <location id="id30" x="-1036" y="-450">
                  <name x="-1054" y="-433">listen</name>
                  <label kind="exponentialrate" x="-1113" y="-
476">normRate</label>
            </location>
            <location id="id31" x="-773" y="-450">
                  <name x="-816" y="-433">turnOffObserver</name>
```

```
                  <label kind="exponentialrate" x="-850" y="-
476">normRate</label>
            </location>
            <location id="id32" x="-518" y="-450">
                  <name x="-510" y="-441">turnOffRescheduler</name>
                  <label kind="exponentialrate" x="-501" y="-
467">normRate</label>
            </location>
            <location id="id33" x="-518" y="-237">
                  <name x="-510" y="-271">turnOffWorkers</name>
                  <label kind="exponentialrate" x="-501" y="-
237">normRate</label>
            </location>
            <location id="id34" x="-773" y="-237">
                  <name x="-808" y="-271">turnOffScout</name>
                  <label kind="exponentialrate" x="-850" y="-
229">normRate</label>
            </location>
            <location id="id35" x="-1036" y="-237">
                  <name x="-1088" y="-272">turnOffMechanic</name>
                  <label kind="exponentialrate" x="-1113" y="-
229">normRate</label>
            </location>
            <location id="id36" x="-1326" y="-238">
                  <name x="-1334" y="-221">end</name>
            </location>
            <init ref="id29"/>
            <transition>
                  <source ref="id35"/>
                  <target ref="id36"/>
                  <label kind="guard" x="-1292" y="-
229">allMechanicsTerminated()</label>
            </transition>
            <transition>
                  <source ref="id34"/>
                  <target ref="id35"/>
                  <label kind="guard" x="-969" y="-229">scoutTerminated</label>
            </transition>
            <transition>
                  <source ref="id33"/>
                  <target ref="id34"/>
                  <label kind="guard" x="-714" y="-
229">allWorkersTerminated()</label>
            </transition>
            <transition>
                  <source ref="id32"/>
                  <target ref="id33"/>
                  <label kind="guard" x="-510" y="-
356">reassignmentTerminated</label>
            </transition>
            <transition>
                  <source ref="id31"/>
                  <target ref="id32"/>
                  <label kind="guard" x="-714" y="-
476">deactivationTerminated</label>
            </transition>
            <transition>
                  <source ref="id30"/>
                  <target ref="id31"/>
                  <label kind="guard" x="-952" y="-476">allCalled()</label>
            </transition>
            <transition>
                  <source ref="id29"/>
                  <target ref="id30"/>
```

```
                  <label kind="assignment" x="-1249" y="-
476">initSession()</label>
            </transition>
            <transition>
                  <source ref="id30"/>
                  <target ref="id30"/>
                  <label kind="select" x="-1071" y="-603">s : station_t</label>
                  <label kind="synchronisation" x="-1062" y="-586">close[s]?
</label>
                  <label kind="assignment" x="-1079" y="-
569">updateCall(s)</label>
                  <nail x="-1079" y="-543"/>
                  <nail x="-994" y="-543"/>
            </transition>
            <transition>
                  <source ref="id31"/>
                  <target ref="id31"/>
                  <label kind="guard" x="-833" y="-586">!
deactivationTerminated</label>
                  <label kind="synchronisation" x="-816" y="-
569">stopDeactivation!</label>
                  <nail x="-816" y="-543"/>
                  <nail x="-731" y="-543"/>
            </transition>
            <transition>
                  <source ref="id32"/>
                  <target ref="id32"/>
                  <label kind="guard" x="-586" y="-586">!
reassignmentTerminated</label>
                  <label kind="synchronisation" x="-569" y="-
569">stopReassignment!</label>
                  <nail x="-561" y="-543"/>
                  <nail x="-476" y="-543"/>
            </transition>
            <transition>
                  <source ref="id33"/>
                  <target ref="id33"/>
                  <label kind="guard" x="-579" y="-135">!
allWorkersTerminated()</label>
                  <label kind="synchronisation" x="-545" y="-119">stopWorker!
</label>
                  <nail x="-561" y="-135"/>
                  <nail x="-476" y="-135"/>
            </transition>
            <transition>
                  <source ref="id34"/>
                  <target ref="id34"/>
                  <label kind="guard" x="-824" y="-135">!scoutTerminated</label>
                  <label kind="synchronisation" x="-800" y="-119">stopScout!
</label>
                  <nail x="-816" y="-135"/>
                  <nail x="-731" y="-135"/>
            </transition>
            <transition>
                  <source ref="id35"/>
                  <target ref="id35"/>
                  <label kind="guard" x="-1122" y="-136">!
allMechanicsTerminated()</label>
                  <label kind="synchronisation" x="-1072" y="-119">stopMechanic!
</label>
                  <nail x="-1079" y="-135"/>
                  <nail x="-994" y="-135"/>
            </transition>
      </template>
```

```
      <template>
            <name>RobotWorker</name>
            <parameter>const worker_t id</parameter>
            <declaration>sector_t    taskX,
            taskY;

territory_t path,
            pathToStation;

error_t     errorType;

int         energyNeeded;

void start()
{
    worker[id].started = true;
}

void initTask()
{
    worker[id].assigned              = true;
    sectorAssigned[workerSector[id]] = true;

    taskX = getX(workerSector[id]);
    taskY = getY(workerSector[id]);
}

void countNeededEnergy()
{
    energyNeeded = path + MAX_DIRT + pathToStation;
}

void countWorkerPath()
{
    path          = countPath(worker[id].posX, worker[id].posY, taskX, taskY);
    pathToStation = countPath(taskX, taskY, station[worker[id].station].posX,
station[worker[id].station].posY);

    countNeededEnergy();
}

void refuseWork()
{
    worker[id].assigned              = false;
    worker[id].refill                = true;
    sectorAssigned[workerSector[id]] = false;
}

void prepare()
{
    worker[id].refill = false;
}

void goToStation()
{
    if (worker[id].posX &gt; station[worker[id].station].posX)
    {
        worker[id].posX = worker[id].posX - 1;
    }
    else if (worker[id].posX &lt; station[worker[id].station].posX)
    {
        worker[id].posX = worker[id].posX + 1;
    }
    else
```

```
    {
        if (worker[id].posY &gt; station[worker[id].station].posY)
        {
            worker[id].posY = worker[id].posY - 1;
        }
        else if (worker[id].posY &lt; station[worker[id].station].posY)
        {
            worker[id].posY = worker[id].posY + 1;
        }
    }

    pathToStation = pathToStation - 1;

    worker[id].energy = worker[id].energy - 1;
}

void goToSector()
{
    if (worker[id].posX &gt; taskX)
    {
        worker[id].posX = worker[id].posX - 1;
    }
    else if (worker[id].posX &lt; taskX)
    {
        worker[id].posX = worker[id].posX + 1;
    }
    else
    {
        if (worker[id].posY &gt; taskY)
        {
            worker[id].posY = worker[id].posY - 1;
        }
        else if (worker[id].posY &lt; taskY)
        {
            worker[id].posY = worker[id].posY + 1;
        }
    }

    path = path - 1;

    worker[id].energy = worker[id].energy - 1;
}

void arriveToSector()
{
    worker[id].posX = taskX;
    worker[id].posY = taskY;
}

void arriveToStation()
{
    worker[id].posX = station[worker[id].station].posX;
    worker[id].posY = station[worker[id].station].posY;
}

void cleanSector()
{
    sectorStatus[taskX][taskY] = sectorStatus[taskX][taskY] - 1;
    worker[id].energy          = worker[id].energy - 1;
}

void loadEnergy()
{
    if (WORKER_ENERGY - worker[id].energy &gt; 5)
```

```
    {
        worker[id].energy = worker[id].energy + 5;
    }
    else
    {
        worker[id].energy = WORKER_ENERGY;
    }

}</declaration>
            <location id="id37" x="-2389" y="-511">
                  <name x="-2406" y="-494">begin</name>
                  <label kind="exponentialrate" x="-2363" y="-
519">normRate</label>
            </location>
            <location id="id38" x="-2185" y="-383">
                  <name x="-2193" y="-366">idle</name>
                  <label kind="exponentialrate" x="-2270" y="-
390">normRate</label>
            </location>
            <location id="id39" x="-1598" y="-382">
                  <name x="-1641" y="-374">error</name>
                  <label kind="exponentialrate" x="-1675" y="-
424">errorRate</label>
            </location>
            <location id="id40" x="-960" y="-383">
                  <name x="-970" y="-417">ride</name>
                  <label kind="exponentialrate" x="-986" y="-
356">normRate</label>
            </location>
            <location id="id41" x="-1598" y="-655">
                  <name x="-1624" y="-689">receive</name>
                  <label kind="exponentialrate" x="-1683" y="-
671">normRate</label>
            </location>
            <location id="id42" x="-1199" y="-587">
                  <name x="-1182" y="-596">decide</name>
                  <label kind="exponentialrate" x="-1292" y="-
594">saveRate</label>
            </location>
            <location id="id43" x="-1598" y="-85">
                  <name x="-1666" y="-85">execute</name>
                  <label kind="exponentialrate" x="-1581" y="-
85">normRate</label>
            </location>
            <location id="id44" x="-2389" y="-222">
                  <name x="-2406" y="-205">end</name>
            </location>
            <location id="id45" x="-1598" y="-748">
                  <name x="-1615" y="-739">return</name>
                  <label kind="exponentialrate" x="-1675" y="-
764">saveRate</label>
            </location>
            <location id="id46" x="-2031" y="-663">
                  <name x="-2031" y="-654">refill</name>
                  <label kind="exponentialrate" x="-2108" y="-
662">saveRate</label>
            </location>
            <init ref="id37"/>
            <transition>
                  <source ref="id46"/>
                  <target ref="id46"/>
                  <label kind="guard" x="-2150" y="-782">worker[id].energy &lt;
WORKER_ENERGY</label>
                  <label kind="assignment" x="-2065" y="-
```

```
765">loadEnergy()</label>
                  <nail x="-1989" y="-739"/>
                  <nail x="-2065" y="-739"/>
            </transition>
            <transition>
                  <source ref="id45"/>
                  <target ref="id45"/>
                  <label kind="guard" x="-1640" y="-867">pathToStation &gt;
0</label>
                  <label kind="assignment" x="-1632" y="-
850">goToStation()</label>
                  <nail x="-1555" y="-825"/>
                  <nail x="-1640" y="-825"/>
            </transition>
            <transition>
                  <source ref="id46"/>
                  <target ref="id38"/>
                  <label kind="guard" x="-2235" y="-586">worker[id].energy ==
WORKER_ENERGY</label>
                  <label kind="assignment" x="-2201" y="-552">prepare()</label>
            </transition>
            <transition>
                  <source ref="id45"/>
                  <target ref="id46"/>
                  <label kind="guard" x="-1878" y="-757">pathToStation &lt;=
0</label>
                  <label kind="assignment" x="-1869" y="-
740">arriveToStation()</label>
            </transition>
            <transition>
                  <source ref="id42"/>
                  <target ref="id45"/>
                  <label kind="guard" x="-1190" y="-663">worker[id].energy &lt;=
energyNeeded</label>
                  <label kind="assignment" x="-1190" y="-
646">refuseWork()</label>
                  <nail x="-1198" y="-672"/>
            </transition>
            <transition>
                  <source ref="id43"/>
                  <target ref="id43"/>
                  <label kind="guard" x="-1675" y="17">worker[id].active
&amp;&amp;
sectorStatus[taskX][taskY] &gt; 0</label>
                  <label kind="assignment" x="-1641"
y="51">cleanSector()</label>
                  <nail x="-1547" y="8"/>
                  <nail x="-1649" y="8"/>
            </transition>
            <transition>
                  <source ref="id40"/>
                  <target ref="id40"/>
                  <label kind="guard" x="-866" y="-409">worker[id].active
&amp;&amp;
path &gt; 0</label>
                  <label kind="assignment" x="-866" y="-
375">goToSector()</label>
                  <nail x="-875" y="-434"/>
                  <nail x="-875" y="-324"/>
            </transition>
            <transition>
                  <source ref="id38"/>
                  <target ref="id44"/>
                  <label kind="guard" x="-2312" y="-
```

```
281">worker[id].active</label>
                  <label kind="synchronisation" x="-2312" y="-264">stopWorker?
</label>
                  <label kind="assignment" x="-2312" y="-
247">worker[id].terminated = true</label>
            </transition>
            <transition>
                  <source ref="id43"/>
                  <target ref="id38"/>
                  <label kind="guard" x="-2082" y="-145">worker[id].active
&amp;&amp;
sectorStatus[taskX][taskY] == 0</label>
                  <label kind="synchronisation" x="-2082" y="-111">done[id]!
</label>
                  <nail x="-1980" y="-153"/>
            </transition>
            <transition>
                  <source ref="id40"/>
                  <target ref="id43"/>
                  <label kind="guard" x="-1156" y="-188">worker[id].active
&amp;&amp;
path &lt;= 0</label>
                  <label kind="assignment" x="-1156" y="-
154">arriveToSector()</label>
                  <nail x="-1207" y="-162"/>
            </transition>
            <transition>
                  <source ref="id42"/>
                  <target ref="id40"/>
                  <label kind="guard" x="-1063" y="-536">worker[id].active
&amp;&amp;
worker[id].energy &gt; energyNeeded</label>
            </transition>
            <transition>
                  <source ref="id39"/>
                  <target ref="id40"/>
                  <label kind="guard" x="-1292" y="-315">worker[id].active
&amp;&amp;
errorType == 2</label>
                  <nail x="-1241" y="-324"/>
            </transition>
            <transition>
                  <source ref="id40"/>
                  <target ref="id39"/>
                  <label kind="guard" x="-1284" y="-434">!
worker[id].active</label>
                  <label kind="assignment" x="-1275" y="-417">errorType =
2</label>
                  <nail x="-1241" y="-451"/>
            </transition>
            <transition>
                  <source ref="id43"/>
                  <target ref="id39"/>
                  <label kind="guard" x="-1768" y="-247">!
worker[id].active</label>
                  <label kind="assignment" x="-1751" y="-230">errorType =
3</label>
                  <nail x="-1649" y="-230"/>
            </transition>
            <transition>
                  <source ref="id39"/>
                  <target ref="id43"/>
                  <label kind="guard" x="-1539" y="-247">worker[id].active
&amp;&amp;
```

```
errorType == 3</label>
                  <nail x="-1547" y="-230"/>
            </transition>
            <transition>
                  <source ref="id39"/>
                  <target ref="id38"/>
                  <label kind="guard" x="-1964" y="-315">worker[id].active
&amp;&amp;
errorType == 0</label>
                  <nail x="-1913" y="-324"/>
            </transition>
            <transition>
                  <source ref="id38"/>
                  <target ref="id39"/>
                  <label kind="guard" x="-1964" y="-426">!
worker[id].active</label>
                  <label kind="assignment" x="-1947" y="-409">errorType =
0</label>
                  <nail x="-1913" y="-443"/>
            </transition>
            <transition>
                  <source ref="id42"/>
                  <target ref="id39"/>
                  <label kind="guard" x="-1292" y="-536">!
worker[id].active</label>
                  <label kind="assignment" x="-1292" y="-519">refuseWork(),
errorType = 0</label>
            </transition>
            <transition>
                  <source ref="id39"/>
                  <target ref="id41"/>
                  <label kind="guard" x="-1539" y="-553">worker[id].active
&amp;&amp;
errorType == 1</label>
                  <nail x="-1547" y="-528"/>
            </transition>
            <transition>
                  <source ref="id41"/>
                  <target ref="id39"/>
                  <label kind="guard" x="-1768" y="-553">!
worker[id].active</label>
                  <label kind="assignment" x="-1751" y="-536">errorType =
1</label>
                  <nail x="-1649" y="-528"/>
            </transition>
            <transition>
                  <source ref="id41"/>
                  <target ref="id42"/>
                  <label kind="guard" x="-1437" y="-
672">worker[id].active</label>
                  <label kind="assignment" x="-1445" y="-
655">countWorkerPath()</label>
            </transition>
            <transition>
                  <source ref="id38"/>
                  <target ref="id41"/>
                  <label kind="guard" x="-1929" y="-
655">worker[id].active</label>
                  <label kind="synchronisation" x="-1929" y="-638">work[id]?
</label>
                  <label kind="assignment" x="-1929" y="-621">initTask()</label>
                  <nail x="-2006" y="-562"/>
            </transition>
            <transition>
```

```
                  <source ref="id37"/>
                  <target ref="id38"/>
                  <label kind="guard" x="-2303" y="-
492">systemCanStart()</label>
                  <label kind="assignment" x="-2286" y="-475">start()</label>
            </transition>
      </template>
      <system>// Place template instantiations here.
scout        = RobotScout();
reassignment = ReassignmentMechanism();
deactivation = DeactivationProcess();
termination  = TerminationMechanism();

// List one or more processes to be composed into a system.
system ControlStation,
       RobotWorker,
       RobotMechanic,
       scout,
       reassignment,
       deactivation,
       termination;
    </system>
      <queries>
            <query>
                  <formula>Pr[&lt;=500; 10000]( &lt;&gt; forall(t : territory_t)
sectorIsCleaned(t))</formula>
                  <comment>5.2.2.1  Tikimybinis sistemos pilno darbo atlikimo
Ä¯vertinimas</comment>
            </query>
            <query>
                  <formula>Pr[&lt;=500; 10000]( &lt;&gt; forall(s : station_t)
ControlStation(s).end)</formula>
                  <comment>5.2.2.2  Tikimybinis visÅ³ valdymo stoÄ‹iÅ³ darbo
pabaigimo Ä¯vertinimas</comment>
            </query>
            <query>
                  <formula>Pr[&lt;=500; 10000] (&lt;&gt; exists(s : station_t)
ControlStation(s).error)</formula>
                  <comment>5.2.2.3  Tikimybinis valdymo stoÄ‹iÅ³ gedimo
galimybÄ…s Ä¯vertinimas</comment>
            </query>
            <query>
                  <formula>simulate [&lt;=500; 250] { r() } </formula>
                  <comment>5.2.2.4  Valdymo stoÄ‹iÅ³ gedimo laikotarpio
simuliavimas</comment>
            </query>
            <query>
                  <formula>Pr[&lt;=500; 10000] (&lt;&gt; exists(w : worker_t)
RobotWorker(w).error)</formula>
                  <comment>5.2.2.5  Tikimybinis robotÅ³-darbininkÅ³ gedimo
galimybÄ…s Ä¯vertinimas</comment>
            </query>
            <query>
                  <formula>Pr[&lt;=500; 10000] (&lt;&gt; (exists(w1 : worker_t)
exists(w2 : worker_t) RobotWorker(w1).error &amp;&amp; RobotWorker(w2).error
&amp;&amp; w1 != w2) )</formula>
                  <comment>5.2.2.6  Tikimybinis keliÅ³ robotÅ³-darbininkÅ³
gedimo galimybÄ…s Ä¯vertinimas</comment>
            </query>
            <query>
                  <formula>Pr( &lt;&gt;[0, 500] ([] [0, 15] (exists(w :
worker_t) RobotWorker(w).error)))</formula>
                  <comment>5.2.2.7  Tikimybinis robotÅ³-darbininkÅ³ gedimo per
nurodytÄ‹ laiko tarpÄ‹ galimybÄ…s Ä¯vertinimas</comment>
```

```
            </query>
            <query>
                  <formula>simulate [&lt;=120; 250] { e() } </formula>
                  <comment>5.2.2.8  RobotÅ³-darbininkÅ³ energijos sunaudojimo
simuliavimas</comment>
            </query>
            <query>
                  <formula>Pr[&lt;=500; 10000] (&lt;&gt; exists exists(s :
station_t) (w : worker_t) ControlStation(s).error &amp;&amp;
RobotWorker(w).error &amp;&amp; worker[w].station == s)</formula>
                  <comment>5.2.2.9  Tikimybinis valdymo stoties ir roboto-
darbininko gedimo galimybÄ▫s Ä¯vertinimas</comment>
            </query>
            <query>
                  <formula>E[&lt;=500; 10000](max: sum(t : territory_t)
sectorStatus[getX(t)][getY(t)] == 0)</formula>
                  <comment>5.2.2.10   IÅ¡valytÅ³ sektoriÅ³ per nurodytÄ¸ laiko
tarpÄ¸ skaitinis Ä¯vertinimas
5.2.2.10.1 Maksimalus skaiÄ¦ius</comment>
            </query>
            <query>
                  <formula>E[&lt;=15;  10000](min: sum(t : territory_t)
sectorStatus[getX(t)][getY(t)] == 0)</formula>
                  <comment>5.2.2.10   IÅ¡valytÅ³ sektoriÅ³ per nurodytÄ¸ laiko
tarpÄ¸ skaitinis Ä¯vertinimas
5.2.2.10.2 Minimalus skaiÄ¦ius</comment>
            </query>
      </queries>
</nta>
```