VILNIAUS UNIVERSITETAS

MATEMATIKOS IR INFORMATIKOS FAKULTETAS

STUDIJŲ PROGRAMA: INFORMATIKA

Paskirstytų sistemų modeliavimas ir verifikavimas remiantis statistinio modelių patikrinimo metodais

Modelling and Verification of Distributed Systems Using Statistical Model Checking Methods

Baigiamasis magistro darbas

Atliko: Nedas Jarmolenka

VU el. p.: nedas.jarmolenka@mif.stud.vu.lt

Darbo vadovas: Prof., Dr. Linas Laibinis

Recenzentas: Dr. Haroldas Giedra

Vilnius

2022

Turinys

Įvadas ... 4
1. Šaltinių analizė ... 5
  1.1 Formalus verifikavimas ... 5
    1.1.1 Modelių tikrinimas ... 6
    1.1.2 Statistinis modelių tikrinimas ... 9
    1.1.3 Laiko automatai ... 10
    1.1.4 Modelių tikrinimo įrankis „*Uppaal*“ ... 11
  1.2 Paskirstytos sistemos ... 17
    1.2.1 Kiberfizinės sistemos ... 17
    1.2.2 Daiktų interneto sistema ir jos modeliavimo pavyzdys ... 18
2. Praktinė dalis ... 24
  2.1 Modeliavimas ... 24
    2.1.1 Aplinkos procesas ... 26
    2.1.2 Temperatūros jutiklio procesas ... 26
    2.1.3 Debesų kompiuterijos valdiklio procesas ... 28
    2.1.4 Oro kondicionieriaus procesas ... 31
    2.1.5 Šildytuvo procesas ... 32
    2.1.6 Taisytojo procesas ... 34
  2.2 Verifikavimas ... 36
Išvados ... 43

Akronimų apibrėžimai

ICT – Informacinės ir komunikacinės technologijos

IEEE – Elektros ir elektronikos inžinerijos institutas

IoT – daiktų internetas (ang. *Internet of things*)

SMC – statistinis modelių tikrinimas (ang. *statistical model checking*)

Įvadas

Šiame darbe nagrinėjamos paskirstytos sistemos ir jų verifikavimas. Tarp svarbesnių tokių sistemų charakteristikų yra išteklių pasidalijimas, lygiagretiškumas, asinchroninis komunikavimas, tolerancija gedimams [1]. Vienos iš sparčiai populiarėjančių paskirstytų sistemų yra kiberfizinės sistemos, kurios pasižymi savo integracija su fiziniu pasauliu. Pavyzdys tokių sistemų yra daiktų internetas (IoT) [3]. Šios sistemos svarbus bruožas yra tai, jog ji turi komponentų, kurie renka informaciją apie fizinį pasaulį arba jį veikia vienokiais ar kitokiais būdais. Šios sistemos vis sudėtingėja ir jų klaidos gali būti kritinės, kitais žodžiais yra atnešančios didelius finansinius nuostolius arba keliančios grėsmę žmonių gyvybėm ar aplinkai, todėl reikalingi griežtesni jų verifikavimo ar validavimo metodai. Tarp tokių metodų yra formalūs metodai, kurie leidžia formaliai verifikuoti sistemą pagal jai iškeltus griežtus reikalavimus [1]. Pavyzdžiui, vienas iš reikalavimų gali būti, jog tam tikro žinomo sistemos sutrikimo atveju sistema turėtų atsistatyti per ne didesnį nei duotą laiką ir veikti toliau [11].

Darbe remiamasi formaliuoju verifikavimu, konkrečiau klasikinio ir statistinio modelių tikrinimo (SMC) metodais. Šiam darbui buvo pasirinktas „*Uppaal*“ įrankis, kuris geba tikrinti sistemų modelius minėtais metodais. Pagrindinis darbo rezultatas yra esamo IoT sistemos, kurios tikslas yra kambario temperatūros reguliavimas, modelio praplėtimas ir verifikavimas esminių reikalavimų atžvilgiu.

Darbo tikslas: atlikti pasirinktos IoT sistemos modeliavimą ir verifikavimą remiantis statistinio modelių patikrinimo metodais.

Darbo tikslui pasiekti iškelti šie **uždaviniai**:

- Atlikus literatūros analizę, apžvelgti modelių tikrinimo metodus ir įrankius, siekiant pagrįsti naudojamų metodų tinkamumą IoT sistemų verifikavimui.
- Atlikti pasirinktos IoT sistemos analizę, identifikuoti jos konfigūracijos ir kitus parametrus, verifikuotinas savybes.
- Naudojantis „*Uppaal*“ įrankiu išplėsti pasirinktos IoT sistemos modelį taip, kad būtų galima jį įvertinti remiantis statistinio modelio patikrinimo (SMC) metodu.
- Verifikuoti identifikuotas IoT sistemos savybes, tokias kaip funkcinio teisingumo (ang. *correctness*), laiko (ang., *timing*), sistemos aklaviečių nebuvimo (ang. *deadlock freeness*), ar invariantines (ang. *invariant*), tikslo pasiekimo (ang., *reachability, liveness*).

- Statistiškai (tikimybiškai) palyginti sistemos veikimo rezultatus, keičiant sistemos konfigūracijos parametrus.

1. Šaltinių analizė

Šiame darbe remiantis įvairiais šaltiniais nagrinėjamas formalus verifikavimas bei jo atšakos – klasikinis modelių patikrinimas, statistinis modelių patikrinimas. Paminima kokiais įrankiais galima atlikti modelių verifikavimą ir pateikta detalesnė įrankio „*Uppaal*“ apžvalga. Nagrinėjamos pasiskirstytos sistemos, kurios glaudžiai siejasi su fiziniu pasauliu (kiberfizinės sistemos, įterptosios sistemos, daiktų internetas) bei jų verifikuotinos savybės. Taip pat apžvelgiamas daiktų interneto sistemos verifikavimo pavyzdys.

1.1 Formalus verifikavimas

Po kompiuterinės ar programų sistemos sukūrimo reikalingas jos verifikavimas, t.y., patikrinimas, ar neatsiranda klaidų jos vykdymo metu. Kai sistema nėra pernelyg sudėtinga ir jos klaidos neatneša didelių pasekmių, dažniausiai tokių sistemų kūrėjai sistemos patikrinimui renkasi testavimo atvejus ar scenarijus (ang. *test cases*), kurie, konkrečių sistemos įvesties ir atitinkamų išvesties duomenų pagrindu, parodo, ar sistema tenkina arba netenkina nurodytą reikalavimą konkrečiu atveju. Tačiau, jeigu sistema yra sudėtinga ar jos klaidos gali būti kritinės, tai reikalingi griežtesni patikrinimo metodai. Minėtas testavimas tam tikrais scenarijais nebetinka, nes matematiniu požiūriu įvesčių skaičius gali artėti prie begalybės ir paprasčiausiai nėra įmanoma patikrinti begalybę įvairių scenarijų.

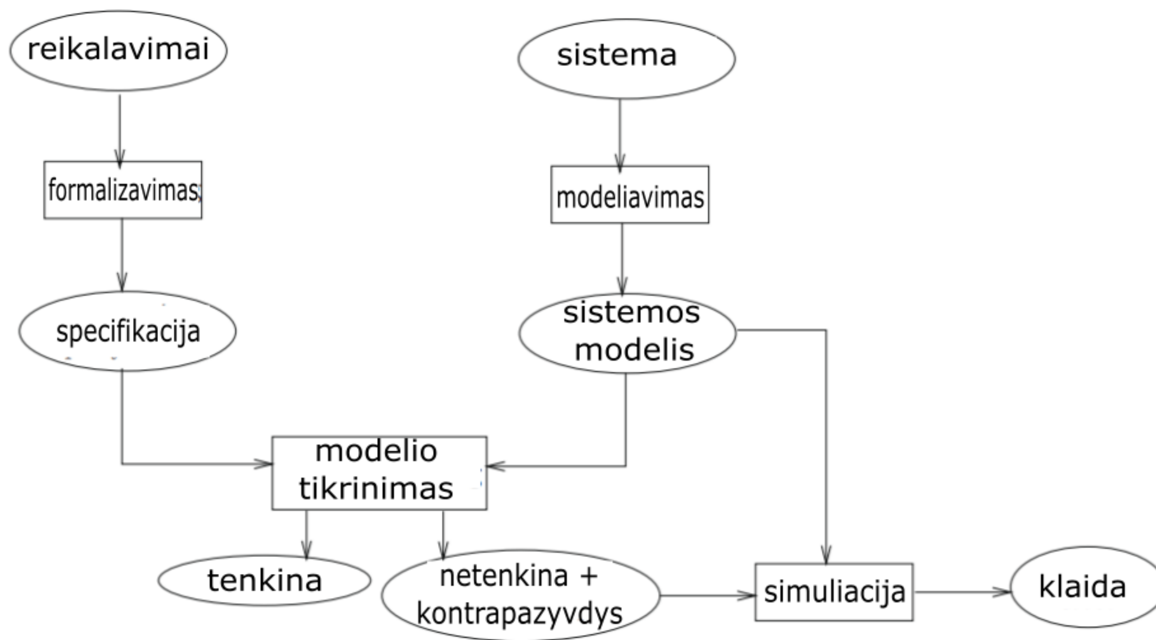
Vienas iš griežtesnių sistemų patikrinimo būdų yra formalus verifikavimas. Formalus verifikavimas yra perspektyvus būdas užtikrinti sistemos saugumo ir kitas kritines savybes, matematiškai įsitikinant, kad sistemos modeliai ar prototipai yra teisingi, naudojant įvairius matematinius ir loginius metodus [4]. Šie metodai yra taip pat naudingi norint kiekybiškai įvertinti sistemos esmines savybes, tokias kaip funkcinis korektiškumas, saugumas, patikimumumas, našumas ir t.t. [5][6], pavyzdžiui, kad sistemos kritinės klaidos tikimybė yra ne didesnė kaip reikalavimuose nurodyta reikšmė.

Formalus verifikavimas skaidosi į du būdus: verifikavimą teoremų įrodymo pagrindu ir modelių tikrinimą. Teoremų įrodymų būdas - tai matematinis modelio savybių įrodymas visiem galimiem sistemos pasikeitimams - reikalauja daugiau žinių, pasiruošimo ir laiko, tačiau duoda didesnę garantiją verifikuojamai sistemai. Tuo tarpu klasikinis modelių tikrinimas yra mechaninis visų sistemos būsenų

patikrinimas remiantis pasiekiamumo grafu [7], sėkmingo veikimo atveju gaunantis greitesnius rezultatus ir leidžiantis patvirtinti reikiamų sistemos savybių galiojimą. Pagrindinis tokio modelio tikrinimo trūkumas yra galimas tikrinamų būsenų skaičiaus "sprogimas" (ang. *state explosion problem*) [9], dėl kurio galima nesulaukti rezultatų.

1.1.1 Modelių tikrinimas

Informatikoje modelių tikrinimas yra metodas, skirtas patikrinti, ar baigtinės būsenos sistemos modelis atitinka nurodytą specifikaciją - reikiamų sistemos savybių rinkinį. Įprastai tai siejama su aparatinės ar programinės įrangos sistemomis, kurių specifikacija sukuriama esamų sistemos reikalavimų pagrindu. Reikalavimuose įprastai yra nurodoma, ką sistema gali arba negali atlikti, kokias savybes turėtų tenkinti. Savybių klasių pavyzdžiai galėtų būti aklaviečių vengimas (ang. *deadlock freeness*), sistemos gyvybingumas ar progresas (ang. *liveness*), saugumas (ang. *safety*), tikslo pasiekimas nepaisant to, ar įvyko gedimai (ang. *fault tolerance*).



1 pav. Modelio tikrinimo schema [7]

Elementari modelio tikrinimo schema pavaizduota 1 pav. Sistemos reikalavimai formalizuojami ir gaunamas savybių rinkinys, o iš sistemos bendro aprašymo yra sukuriamas sistemos modelis, naudojantis viena iš egzistuojančių modeliavimo kalbų. Dažnai šios kalbos yra pagrįstos griežta matematine ir logine semantika. Modelio tikrinimo metu perrinkimo būdu pereinant per visas galimas sistemos modelio būsenas, tikrinama, ar tenkinamos savybės iš formalizuotų savybių rinkinio. Jei

pereinant per visas sistemos būsenas savybė patvirtinama, tai grąžinamas teigiamas rezultatas, kad sistemos modelis tenkina savybę. Jeigu sistemos modelyje atsiranda būsena, kuri netenkina savybės, grąžinimas neigiamas rezultatas ir kontrapavyzdys (ang. *counterexample*), kuris parodo kaip sistemos modelis gali pasiekti būseną, kurioje savybė yra netenkinama. Simuliacijos metu stebimas sistemos modelio veikimas. Jeigu sistemos modelyje yra klaidų, pavyzdžiui, modelio aprašyme yra kalbos sintaksės klaidų, simuliatorius grąžina klaidą. Kai modelyje nėra klaidų, simuliatoriaus pagalba galima stebėti, kaip keičiasi sistemos būsenos, o kai reikia, iš naujo paleisti simuliaciją, keisti vykdymo greiti arba sustabdyti simuliaciją. Šie veiksmai leidžia detaliau analizuoti sistemos modelį ir atlikti modelio koregavimus, jeigu analizės metu pastebėta, kad modelis turi trūkumų, neapibrėžtumų. [7]

Modelių tikrinimas yra gana plačiai taikomas ICT sistemose. Pavyzdžiui, tokiu būdu buvo aptiktos aklavietės skrydžių bilietų rezervacijų sistemose, buvo verifikuoti modernūs elektroniniai, komunikacijos protokolai, bei keletas IEEE standartų, kas leido pagerinti namų buities technikos sistemų našumą. Keletas nepastebėtų klaidų buvo atrasta verifikuojant „Deep Space 1“ erdvėlaivio vykdymą. [7]. Taigi verifikavimas modelių tikrinimo būdu tinkamas ne tik kritinių klaidų paieškai, bet ir sistemų našumui pajėgumui gerinti.

Modelių tikrinimas skirstomas į etapus:

- Modeliavimo etapas, kurio metu aprašomas modelis bei formalizuojami reikalavimai;
- Simuliacijos etapas, kurio metu vykdomas sistemos modelis, stebimi vykdymo žingsniai, modelis koreguojamas (jei randama klaidų);
- Verifikavimo etapas, kurio metu analizuojama, ar sistemos modelis tenkina savybes gautas iš reikalavimų. Jeigu yra savybių, kurių netenkina verifikuojama sistema, analizuojami kontrapavyzdžiai, daromos išvados, keičiama sistemos struktūra bei jos modelis ir verifikavimo etapas kartojamas. Šiame etape pasitaiko atvejų, kuomet verifikavimas gali užtrukti labai ilgai dėl didelio sistemos modelio būsenų skaičiaus. Tokiais atvejais didinamas kompiuterio, kuris atlieka verifikavimą, pajėgumas, priskiriant daugiau resursų skaičiavimams.

Kaip buvo minėta, savybės turėtų būti apibrėžtos tiksliai ir nedviprasmiškai. Tam dažniausiai naudojamos specifinės modeliavimo kalbos ar logikos, pavyzdžiui, tiesinė laiko logika (ang. *Linear*

Temporal Logic), skaičiavimo medžio logika (ang. *Computation Tree Logic*) [7]. Praktika parodė, kad šios logikos yra tinkamos ICT sistemų savybių aprašymui. Iš esmės, laiko logika yra tradicinės teiginių logikos išplėtimas operatoriais, kurie leidžia išreikšti sistemos elgesį laike ar sekoje veiksmų. Laiko logikos leidžia išreikšti įvairių klasių savybes, tokias kaip funkcinio teisingumo (ang. *functional correctness*), kuri susijusi su tuo, ar sistema atlieka tai, ką turėtų atlikti. Taip pat pasiekiamumo (ang. *reachability*), kurios pavyzdys būtų, ar sistemą nepasieks užstingimo būsenos. Dar yra saugumo (ang. *safety*) (kad kažkas tikrai neįvyks), realaus laiko (ang. *real-time*) (ar sistema teisingai elgiasi laike) [7].

Modelių tikrinimo privalumai:

- Pritaikomas plačiam spektrui sistemų, tokių kaip įterptinių sistemų, programų sistemų, aparatinės įrangos sistemų verifikavimui.
- Palaikomas dalinis verifikavimas, t.y. savybės gali būti tikrinamos individualiai.
- Pateikiama papildoma informacija, kai verifikuojant gaunama, kad sistema netenkina konkrečios savybės (kontrapavyzdžių ir veiksmų sekų, vedančių prie klaidos, pavidalu).
- Tyrėjui, kuris atlieka modeliavimą ir verifikavimą, nėra būtinas ypatingai aukštas žinių lygis.
- Šis verifikavimas gali būti nesudėtingai integruojamas į sistemos kūrimo procesą. Tyrimai rodo, kad net gali patrumpinti sistemos kūrimo laiką.
- Turi griežtą matematinį pagrindą: remiasi grafų teorijos algoritmais, duomenų struktūromis bei logika.

Tačiau modelių tikrinimas turi ir trūkumų:

- Labiau tinka sistemoms, kuriose dominuoja operacijos (ang. *control-intensive*), o ne duomenų kiekis (ang. *data-intensive*).
- Kadangi modelio tikrinimo vykdymo metu vyksta perrinkimas per visas galimas būsenas, tai jeigu būsenų skaičius labai išauga, reikalinga didelio našumo ir didelės apimties aparatinė įranga, o kartais net naudojant didelio našumo kompiuterius užtrunka daug laiko ar išvis nepavyksta gauti rezultatų. Šis reiškinys žinomas kaip būsenų skaičiaus „sprogimas" (ang. *state explosion problem*) [9].

- Negalima verifikuoti sistemų, kurios potencialiai turi begalybę skirtingų būsenų.
- Modelių tikrinimas verifikuoja sistemos modelį (prototipą), bet ne pačią sistemą, todėl gaunamų rezultatų teisingumas ir tikslumas priklauso nuo to, kiek sistemos modelis atitinka realią sistemą.
- Verifikuojamos tik pateiktos savybės reikalavimuose. Nėra garantijos, kad sistemai galios savybės, kurios reikalavimuose nėra pateiktos.
- Modelių tikrinimo įrankis (programinė įranga), taip pat gali turėti defektų ir grąžinti klaidingus rezultatus [7].

1.1.2 Statistinis modelių tikrinimas

Statistinis modelio tikrinimo metodas yra modelių tikrinimo metodo atmaina, kuri leidžia verifikuoti sistemą, neperrenkant visų galimų sistemos būsenų, ir grąžinti rezultatą su tam tikru patikimumu (tikimybe) [10], tuo pačiu statistiškai ar tikimybiškai įvertinant sistemos savybių galiojimą. Šis būdas yra klasikinio modelių tikrinimo papildymas ir toks metodas yra ypatingai tinkamas stochastinių sistemų analizei.

Tarkime turime stochastinę sistemą S ir savybę ϕ, kurios atžvilgiu norime verifikuoti sistemą. Statistinis modelių tikrinimo metodas gali atsakyti į dviejų tipų klausimus:

- Kiekybinį;
- Kokybinį.

Kiekybinis kelia klausimą, kokia tikimybė, kad sistema S tenkina savybę ϕ. Atsakymas gali būti apskaičiuojamas remiantis tikimybių įverčių metodais, kurie remiasi *Chernoff-Hoeffding* riba. Tai yra tikimybės p (tikrosios tikimybės, kad sistema S tenkina savybę ϕ) įverčio p' skaičiavimas, kai

$$Pr(\,|\,p' - p\,|\, < \epsilon) \geq 1 - \alpha \quad (1)$$

kur ϵ – tikslumas, o α – patikimumo dydis (ang. *confidence level*) [12] [13].

Tuo tarpu kokybinis kelia klausimą, ar tikimybė, kad sistema S tenkina savybę ϕ, yra didesnė už tam tikrą vertę (ang. *threshold*) θ. Tokio tipo klausimai dažniausiai atsakomi hipotezių tikrinimo būdu. Jei tariame, kad p yra tikimybė, kad S tenkina savybę ϕ, tai hipotezių tikrinimo būdas sprendžia, kuri iš hipotezių $H_0 : p \geq \theta$ ir $H_1 : p < \theta$ yra teisinga, kai duotas patikimumo dydis (α, β) [14] [15].

1.1.3 Laiko automatai

Kita matematinė teorija, kuri pagrindžia paskirstytų sistemų formalų verifikavimą, yra laiko automatai. Laiko automatų teorija gali būti naudojama sistemų elgesiui laike modeliavimui ir tyrimui. Laiko automatas A formaliai gali būti užrašomas taip:

$$A = \langle N, l_o, \Sigma, X, E, I \rangle \quad (2)$$

kur:

- N – baigtinė lokacijų (automato būsenų) aibė;
- l_o – pradinių lokacijų aibė, $l_o \subseteq N$;
- Σ – automato veiksmų arba įvykių aibė. Elementas a iš Σ gali reprezentuoti veiksmą, dėl kurio įvyksta perėjimas iš vienos lokacijos į kitą;
- X – baigtinė automato laikmačių aibė;
- $E \subseteq N \times B(X) \times \Sigma \times 2^X \times N$ yra laiko automato perėjimų aibė. Kiekvienas $\langle l, g, a, r, l' \rangle \in E$ reprezentuoja pasikeitimą nuo l lokacijos iki l' lokacijos. Šį būsenos pasikeitimą galima žymėti $l \xrightarrow{g,a,r} l'$. Čia $r \subseteq X$ yra kintamieji, kurie turi būti atstatyti į pradines reikšmes arba gauti naujas reikšmes būsenos pasikeitimo metu. g reprezentuoja laiko apribojimus, kurie turi būti tenkinami, kad įvyktų būsenos pasikeitimas. $a \in \Sigma$ reprezentuoja sinchronizaciją su kitais sistemos komponentais (automatais) būsenos pasikeitimo metu.

- $I := N \rightarrow B(X)$ yra lokacijų aibės sąryšis su laiko apribojimų (taip vadinamų laiko invariantų) aibe. [16]

Laiko automato būsena laike taip pat turi apibrėžimą. Laiko automato būsenos laike apibrėžimas yra formuluojamas kaip $\langle l, u \rangle$, kur $l \in N$ yra lokacija, o u yra laikrodžio kintamojo (iš aibės X) reikšmė. [16]

Kiekvienai lokacijai $l \in N$, įskaitant ir pradinę lokaciją l_o yra galimi dviejų tipų būsenų pasikeitimai:

1) $\langle l, u \rangle \xrightarrow{d} \langle l, u + d \rangle$, kur $l \in N, u \in I(l), d \in R^+$ ir $u + d \in I(l)$. Tokio tipo pasikeitimas yra vadinamas laiko pasikeitimu.

2) $\langle l, u \rangle \xrightarrow{g,a,r} \langle l', u' \rangle$, jei egzistuoja $l \xrightarrow{g,a,r} l'$, kur $l, l' \in N,\ u \in I(l), u \in g,\ u' = [r \mapsto c]u\ ir\ u' \in I(l')$, čia $[r \mapsto c]u$ reprezentuoja laikrodžio ir kitų kintamųjų reikšmių priskyrimus, įskaitant

laikrodžio vertės nustatymą į nulį. Tokio tipo pasikeitimas yra vadinamas lokacijos pasikeitimu.

Sistemos užstingimas (ang. *deadlock*) taip pat turi apibrėžimą laiko automato kontekste. Jeigu turime laiko automatą būsenoje $\langle l, u \rangle$ ir būsenos pasikeitimo sąlyga $\langle l, u \rangle \xrightarrow{g,a,r} \langle l', u' \rangle$ visada nėra tenkinama kintant laikui $\langle l, u \rangle \xrightarrow{d} \langle l, u + d \rangle$, tai laikoma, jog laiko automatas yra užstingęs (ang. *deadlock*) laikrodžio *u* atžvilgiu. [16]

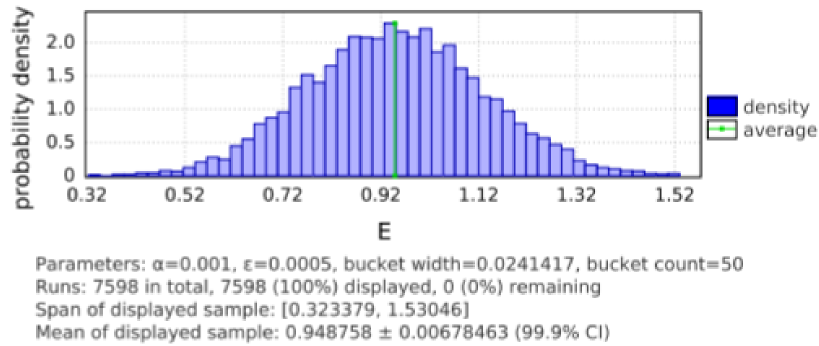
Laiko automatai gali sudaryti tinklą. Jeigu turime laiko automatus $A_1 = \langle N_1, l_o, \Sigma_1, X_1, E_1, I_1 \rangle$ ir

$A_2 = \langle N_2, l_{2o}, \Sigma_2, X_2, E_2, I_2 \rangle$ ir šių laiko automatų veiksmų aibės tenkina sąlygą $\Sigma_1 \cap \Sigma_2 \neq \emptyset$, tai A_1ir A_2 gali būti sujungti ir žymimi $A_1||A_2$. Tai galima laikyti nauju laiko automatu ir žymėti $A_1||A_2 = \langle N_1 \cup N_2, \langle l_{10}, l_{20} \rangle, \Sigma_1 \cup \Sigma_2, X_1 \cup X_2, E_1 \cup E_2, I_1 \cup I_2 \rangle$. Naujam laiko automatui A_i, jeigu tenkinama sąlyga $(\Sigma_1 \cup \Sigma_2) \cap \Sigma_i \neq \emptyset$, galimas dar vieno automato A_i prijungimas. Kai du arba daugiau laiko automatų yra sujungti, tai $M = A1 \mid\mid A2 \mid\mid \ldots \mid\mid An \mid n \geq 2$ vadinamas laiko automatų tinklu.

Modeliuojant paskirstytas sistemas, dažniausiai vienas laiko automatas laikomas vienu sistemos komponentu, o visa sistema laikoma laiko automatų tinklu. Yra nemažai modeliavimo ir verifikavimo įrankių, kuriais galima modeliuoti sistemą laiko automatų principu. Apie tai plačiau aprašoma kitame skyriuje.

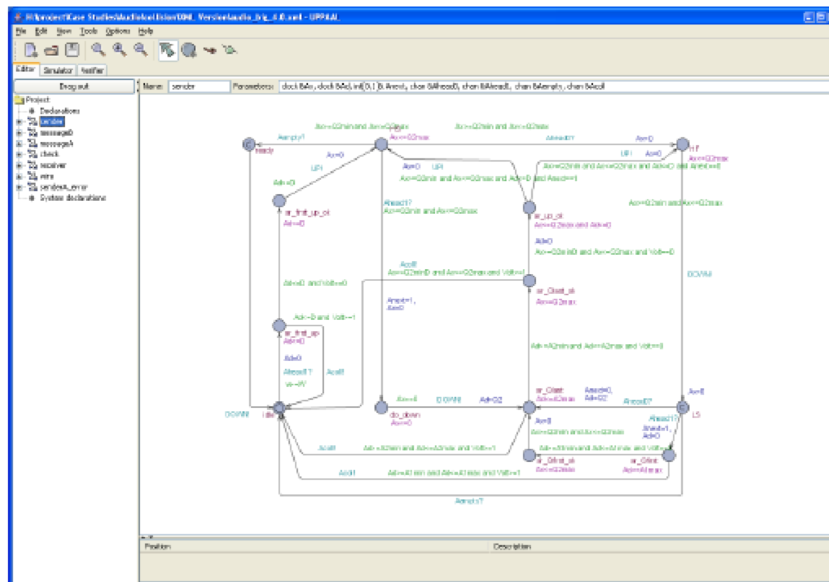
1.1.4 Modelių tikrinimo įrankis „*Uppaal*“

Egzistuoja sukurti įrankiai, kurie gali verifikuoti sistemą naudojant minėtus klasikinį ir statistinį modelių patikrinimo metodus. Tokių įrankių pavyzdžiai yra „*Uppaal*“, „*Prism*“ , „*Spin*“, „*NuSMV*“ ir t.t. Šiame skyriuje dėmesys bus skiriamas „*Uppaa*“ įrankiui. „*Uppaal*“ buvo sukurtas Uppsala ir Aalborg universitų mokslininkų [8] ir yra skirtas sistemoms, kurias galima modeliuoti naudojant nedeterministiškus (ang. *non-deterministic*) procesus su baigtinėmis valdymo struktūromis (ang. *finite control structures*) ir realaus laiko laikrodžiais, verifikuoti [4]. „*Uppaal*“ modeliavimo kalbos teorinis pagrindas yra laiko automatų (ang. *timed automata*) teorija, trumpai aprašyta praeitame skyrelyje.

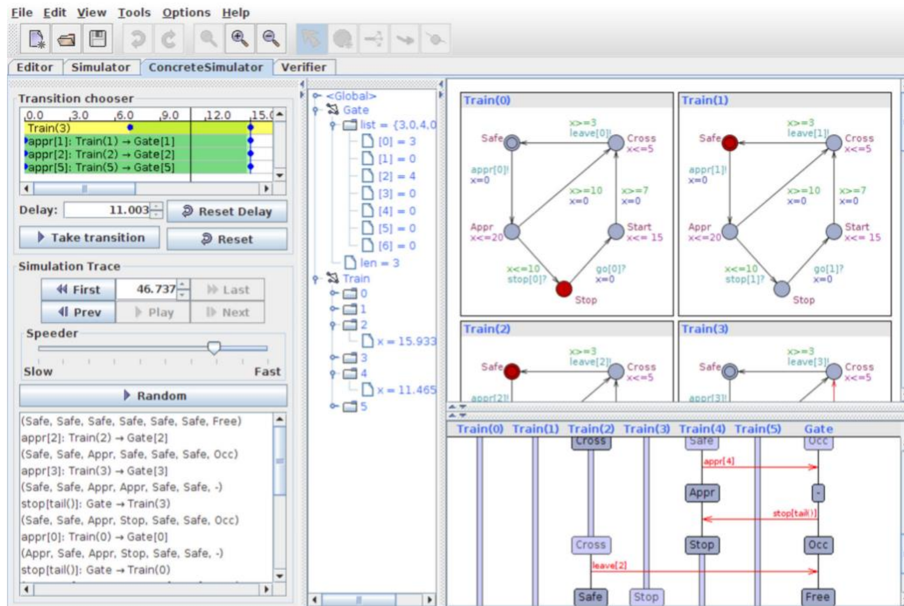


1 pav. „*Uppaal*“ įrankiu gautas tikimybės tankio pasiskirstymas [17]

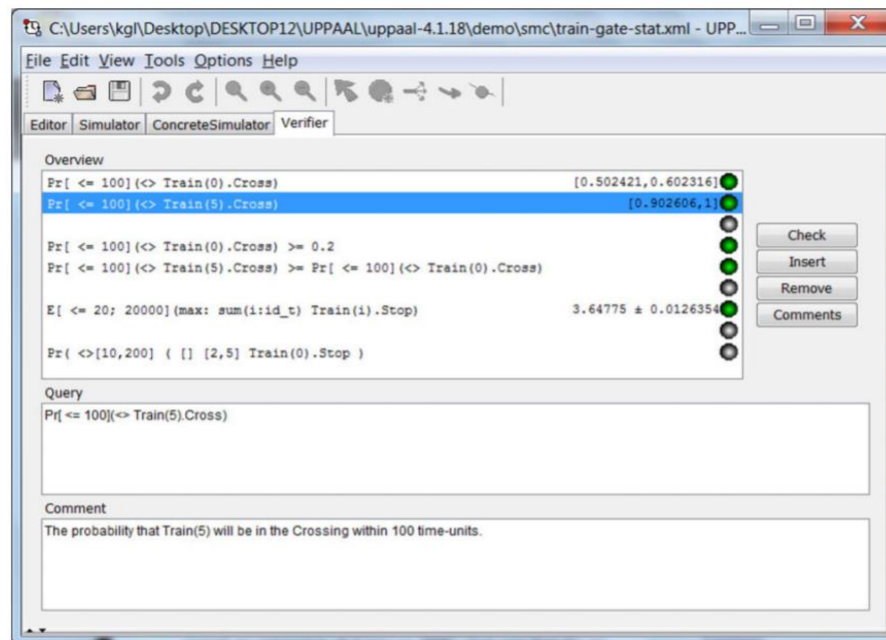
„*Uppaal*“ įrankis palaiko ne tik klasikinį, bet ir statistinį modelio tikrinimo metodą, kuris leidžia patikrinti ne tik tai, ar sistema tenkina arba netenkina tam tikrą reikalavimą, bet ir grąžina statistinį įvertį, kiek vienoks ar kitoks rezultatas yra tikėtinas. Tai leidžia daryti tyrimus, keičiant sistemos parametrus ir lyginant gautus statistinius įverčius (rezultato pavyzdys pateiktas 1 pav.), darant išvadas apie modeliuojamą sistemą. Pavyzdžiui, kiberfizinės sistemos atveju galima nagrinėti, kaip didėja arba mažėja sistemos atsistatymo laikas didinant arba mažinant galimų sutrikimų skaičių ar jų tikimybę.



2 pav. „*Uppaal*“ įrankio grafinė sąsaja (modelio aprašymo posistemė) [8]



3 pav. „*Uppaal*“ įrankio grafinė sąsaja (simuliatoriaus posistemė) [17]



4 pav. „*Uppaal*“ įrankio grafinė sąsaja (modelio tikrinimo posistemė) [17]

Įrankis susideda iš trijų dalių:

1) Modelio aprašymo posistemė. Jos grafinė sąsaja yra pateikta 2 pav. Šios posistemės pagalba yra aprašoma sistema, jos pradinė būsena, jos visos įmanomos, pasiekiamos būsenos, būsenų pasikeitimai, laikrodžiai, kurie matuoja laiką, sistemos komponentės ar procesai ir jų sinchronizavimas, duomenų kintamieji.

2) Simuliavimo posistemė. Ši posistemė leidžia analizuoti modelį įvairiuose vykdymo scenarijuose, pasitikrinti, ar nėra akivaizdžių kritinių klaidų prieš verifikuojant sistemą. Grafinė sąsaja yra pateikta 3 pav.

3) Verifikavimo posistemė. Ši sistemos dalis leidžia verikuoti modelį statistinio modelio patikrinimo metodu. Jos grafinė sąsaja yra pavaizduota 4 pav. Būtent šioje posistemėje verifikuojamos tokios sistemos savybės, kaip pasiekiamumo (ang. *reachability*), laiko (ang. *timing*), aklaviečių nebuvimo (ang. *deadlock freeness*), invariantines (ang. *invariant*), išreikštos kaip laiko logikos (ang. *temporal logic*) formulės.

Sistemos modelis susideda iš grupės komponenčių arba procesų. Kiekvienas toks procesas aprašomas, nurodant jo lokacijas (ang. *locations*), kurios atspindi proceso atitinkamas būsenas. Lokacijos yra sujungtos perėjimais (ang. *edges*, *transitions*). Lokacijos yra skirstomos į tipus:

- Pradinė (ang. *initial*) - kiekvienas modelis turi vieną pradinę lokaciją. Ši lokacija laikoma proceso pradžia;
- Paprasta lokacija - būsena, kurioje sistema gali būti kurį laiką (paprastai apribotą laiko invariantu);
- Skubi (ang. *Urgent*) - laikas negali eiti, kai procesas yra skubioje lokacijoje, pavyzdžiui, sistema turi neužtrukdama pakeisti lokaciją vienu iš įmanomų perėjimų;
- Įsipareigojusi (ang. *commited*) - veikia panašiai kaip skubi lokacija, bet naudojama, kai reikalingas griežtesnis atsižvelgimas į sinchronizacija su kitais procesais.

Lokacijose galima apibrėžti laiko invariantą arba eksponentinį dažnį, kurie apriboja buvimo lokacije laiką. Laiko invariantas nurodo sąlygas, išreikštas laikrodžių apribojimais arba skirtumais. Eksponentiniai dažniai nurodo lokacijos tikimybinį uždelsimą. Perėjimai tarp lokacijų gali būti susieti su kintamųjų sąlygomis ar predikatais (ang. *guard*), sinchronizacija su kitais procesais (ang. *synchronisation*) ir kintamųjų reikšmių atnaujinimais. Jei perėjimo sąlygos išraiška yra teisingos, tai perėjimas yra galimas. Atskirų modelio procesų perėjimai gali sinchronizuotis kanalų (ang. *channels*) pagalba. Dar gali būti apibrėžti perėjimų tikimybiniai svoriai (ang. *probability weights*), kuriais modeliuojama perėjimo tikimybę.

Verifikavimo posistemėje pateikiamos užklausos sistemos verifikavimui. Šios užklausos yra suformuluotos naudojantis laiko logikos, o konkrečiau TCTL (ang. *Timed Computation Tree Logic*),

sintakse. Pavyzdžiui, įrankis gali priimti užklausas tokių pavidalų: *A*[]*p*, *A*<>*p*, *E*<>*p*, *E*[]*p* ir *p* --> *q*. Čia *p* ir *q* yra tikrinamos savybės. Operatorius *A* reiškia savybės *p* galiojimą visuose sistemos vykdymo keliuose, o operatorius *E* – bent vieną sistemos vykdymo kelią, kuriame galioja savybė *p*. Žymėjimas [] atitinka laiko logikos operatorių □, kuris reiškia visada (ang. *always*). Žymėjimas <> atitinka laiko logikos operatorių ◊, kuris reiškia ilgainiui laike (ang. *eventually)*. Taigi užklausa *A*[]*p* tikrinama, ar kiekviename sistemos vykdymo kelyje visada galioja savybė *p*. Užklausa *A*<>*p* tikrinama, ar kiekviename sistemos vykdymo kelyje ilgainiui galios savybė *p*. Užklausa *E*<>*p* tikrinama, ar bent viename sistemos vykdymo kelyje visada galios savybė *p*. Užklausa *E*[]*p* tikrinama, ar bent viename sistemos vykdymo kelyje ilgainiui galios savybė *p.* Operatorius --> atitinka loginę implikaciją →, tačiau tikrinamą skirtingose sistemos būsenose viename iš galimų vykdymo kelių. Kitais žodžiais, užklausa *p* --> *q* tikrinama, jei bent viename sistemos vykdymo kelyje galioja savybė *p,* tai neišvengiamai tame pačiame vykdymo kelyje turi galioti savybė *q*.

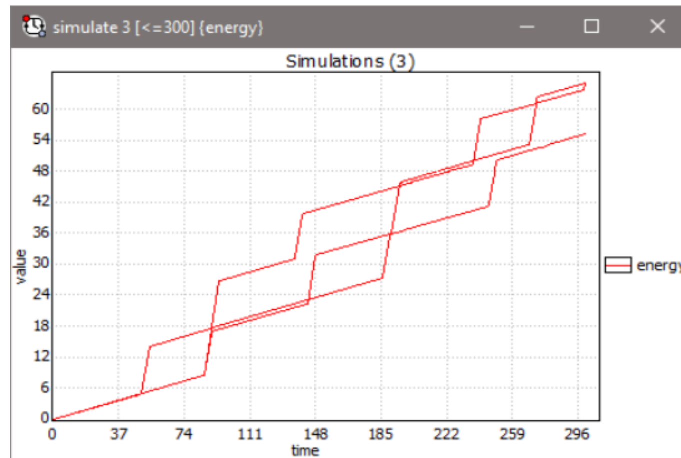
Be to, SMC papildymas leidžia pateikti kitokių užklausų ir sugrąžinti rezultatus grafiniu formatu. Pavyzdžiui, jeigu reikia pasižiūrėti būsenų arba kintamųjų *p, .., q* kitimą laike simuliacijos metu, galima pateikti užklausą:

simulate N [< = bound] {p, .., q}.

Čia *N* – natūralus skaičius, kuris reiškia atliekamų simuliacijų skaičių. <= žymima nelygybė ≤, o *bound* yra laiko ribos vertė. *p, .., q* yra būsenų, kintamųjų išraiškos. Pavyzdžiui, užklausa:

simulate 3 [<=300] {energy}

užklausia kintamojo *energy* reikšmių nuo simuliacijos pradžios iki 300 laiko vienetų. Simuliacija kartojama *N=3* kartus. Įrankio grąžinamas rezultatas galėtų būti toks, koks pavaizduotas 5 pav.

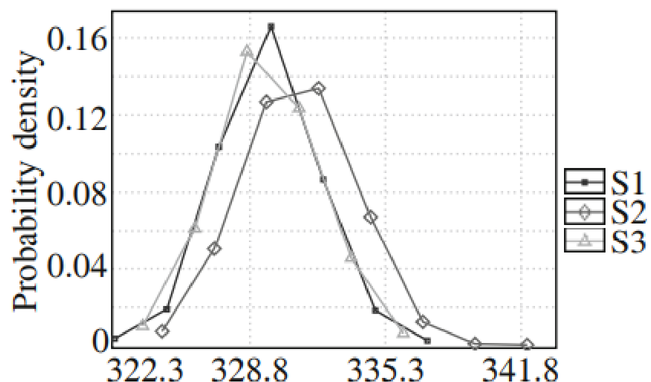


5 pav. „*Uppaal*“ įrankio užklausos *simulate 3 [<=300] {energy}* pavyzdinis rezultatas [18]

Tikimybės pasiskirstymą galima užklausti užklausa:

Pr[<= bound]{<> p}

Čia *bound* yra laiko ribos vertė. *p* yra sistemos būsenos arba kintamojo išraiška. Pavyzdžiui, [19] buvo daromas tyrimas su kiberfizinėm sistemom ir energijos suvartojimu pastatuose. Kitais žodžiais, autoriai pateikė užklausa *Pr[Monitor.energy<=1000000](<> time>=2*day)*. Rezultato grafikas yra pateiktas 6 pav.



6 pav. Tikimybiniai skirstiniai gauti statistinio modelių tikrinimo įrankio pagalba [19]

Taip pat galima tikrinti ar tikimybė yra didesnė arba mažesnė už konkrečia reikšmę su tokia užklausa:

Pr[<=bound](<>p) >= p_o

Čia *bound* yra laiko ribos vertė. *p* yra sistemos tikrinamos būsenos arba kintamojo išraiška. p_o skaičius nuo 0 iki 1. Pavyzdžiui, [20] autoriai naudoja pateikia užklausą:

Pr[<=20](<> time >=12 and y>=4) >= 0.45

Šia užklausa tikrinama, ar tikimybė, jog kintamasis *time* bus didesnis arba lygus *12* ir kintamasis *y* bus didesnis arba lygus *4*, bus didesnė arba lygi *0,45*, kai vykdymo laikas yra nuo *0* iki *20*. Įrankis leidžia lyginti tikimybę ne tik su skaičiumi, bet ir kita tikimybe, pavyzdžiui, tokia išraiška:

$$Pr[<=bound_2](<>p_1) >= Pr[<=bound_1](<>p_2)$$

Čia $bound_1$ ir $bound_2$ yra laiko ribos vertės, o p_1 *ir* p_2 yra sistemos tikrinamų būsenų arba kintamųjų išraiškos.

1.2 Paskirstytos sistemos

Paskirstyta sistema yra sistema, kuri sudaryta iš tinkle sujungtų autonominių kompiuterių bei atitinkamai įdiegtos paskirstytos programinės įrangos. Kompiuteriniai tinklai suteikia reikalingas sąlygas sistemos komponentų tarpusavio komunikacijai. Vienos iš svarbesnių tokios sistemos charakteristikų yra išteklių pasidalijimas, lygiagretiškumas, asinchroninis komunikavimas, tolerancija gedimams [1].

1.2.1 Kiberfizinės sistemos

Vienos iš sparčiai populiarėjančių paskirstytų sistemų yra sistemos, kurios persipina su fiziniu pasauliu. Tarp tokių sistemų yra kiberfizinės sistemos (ang. *Cyber-physical systems*) [2], įterptosios sistemos (ang. *Embedded systems*), daiktų internetas (ang. *Internet of things*) [3]. Jų svarbus bruožas yra tai, jog jos turi komponentų, kurie renka informaciją apie fizinį pasaulį arba jį veikia vienokiais ar kitokiais būdais, pavyzdžiui, per valdomus prietaisus keičiama aplinkos temperatūra. Kadangi šios sistemos vis sudėtingėja ir jų klaidos gali būti kritinės, o tai yra atnešančios didelius finansinius nuostolius arba keliančios grėsmę žmonių gyvybėm ar aplinkai, reikalingi griežtesni jų verifikavimo ar validavimo metodai. Tarp tokių metodų yra ankstesniuose skyriuose minėti formalūs metodai, kurie leidžia formaliai verifikuoti sistemą pagal jai iškeltus griežtus reikalavimus [1]. Pavyzdžiui, keli reikalavimai gali būti, jog tam tikro žinomo sistemos sutrikimo atveju sistema turėtų atsistatyti per ne didesnį nei duotą laiką ir veikti toliau [11]. t.y.pasiekiamumas (ang. *reachability*), ar neįvyks kažkurio komponento sistemoje sugedimas (ang. *safety*), ar sistema geba laiku atlikti užduotis. Yra atveju, kuomet tokios sistemos yra verifikuojamos, pavyzdžiui, [21] buvo verifikuota kuro kontrolės sistema, kuri sudaryta iš jutiklių. Pati sistema palaiko toleravimą jutiklių gedimams. [22] buvo verifikuotas kiberfizinės sistemos saugumas esant retiems įvykiams. Kiberfzinės sistemos verifikavimo pavyzdžių taip pat galima rasti [23], [24], [25].

1.2.2 Daiktų interneto sistema ir jos modeliavimo pavyzdys

Kaip buvo minėta ankstesniame poskyryje, daiktų internetas yra paskirstyta sistema, kuri yra glaudžiai susijusi su fiziniu pasauliu. Ji yra dažniausiai sudaryta iš jutiklių ir valdiklių, kurie tarpusavyje komunikuoja radijo bangomis, interneto protokolu. Jutikliai gali rinkti informaciją apie fizinę aplinką, o valdikliai veikti aplinką, kai valdikliams atsiunčiama komanda. Kadangi šios sistemos yra dinamiškos, jų resursai riboti, pasižymi atvirumu (ang. openness), tai iškyla nemažai iššūkių kuriant tokias sistemas. Todėl net ankstyvame tokių sistemų kūrimo etape dažnai pasitelkiama į formaliu verifikavimu, modelių tikrinimą griežtais matematiniais metodais, kad būtų užtikrinta, jog daiktų interneto sistema atitinka funkcionalumo ir patikimumo reikalavimus. Kadangi ši sistema yra fiziniame pasaulyje, kupiname kintančių procesų ir neapibrėžtumų, daiktų interneto sistemoje dažniausiai svarbiausi atributai yra einamoji būsena ir laikas. Dėl šios priežasties sistema gali būti modeliuojama naudojantis laiko automatu (ang. *timed automata*), kurie aprašo ir atspindi sistemos elgesį laike, fizinės aplinkos, įrenginių laiko apribojimus, būsenų pasikeitimus. Šiame skyriuje bus apžvelgiamas daiktų interneto sistemos modeliavimo ir verifikavimo klasikiniu modelio tikrinimo būdu pavyzdys aprašytas [16] šaltinyje.

Daiktų interneto sistemos modeliavimas remiantis laiko automatų teorija gali būti atliekamas dviem žingsniais:

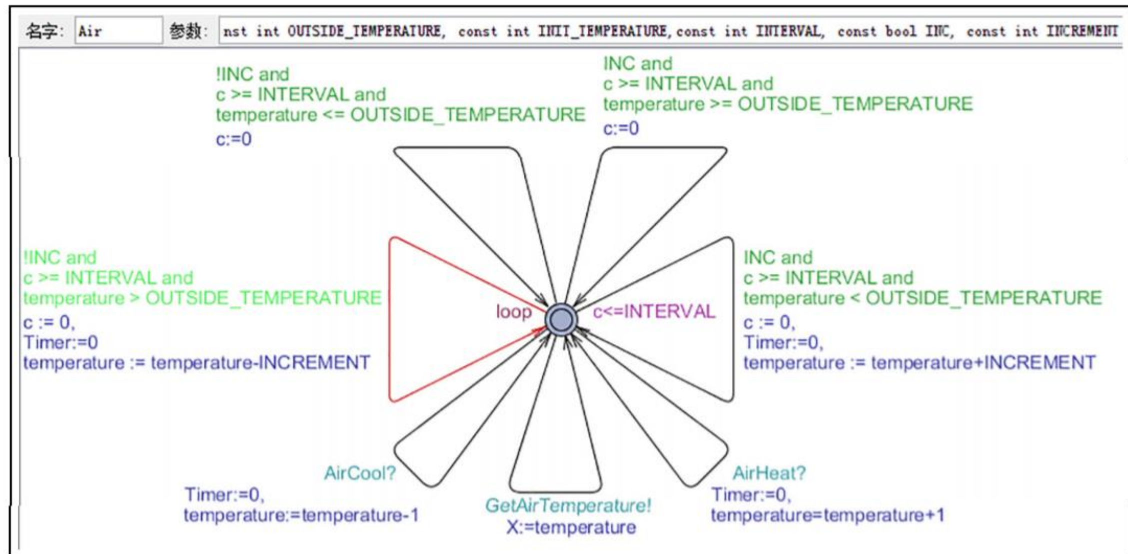
1) Kiekvienas sistemos komponentas modeliuojamas analizuojant jo paskirtį ir tikslą, atitinkamai užpildant laiko automato apibrėžimo parametrus. Taip gaunamas komponento modelis laiko automato pagrindu.
2) Daiktų interneto komponentų laiko automato modeliai formuoja tinklą tarpusavyje sinhronizuodami savo veiksmus (pranešimų ar komukavimo kanalų pagalba). Pabaigoje gaunamas viso daiktų interneto sistemos laiko automatų tinklo modelis.

Modeliuojant daiktų interneto sistemą laiko automatais yra modeliuojami tiesiog daiktų interneto komponentai (jutikliai, valdikliai). Be fizinių prietaisų taip pat gali būti ir virtualios duomenų apdorojimo paslaugos (ang. *service*), pavyzdžiui, debesų kompiuterija. Net pati fizinė aplinka, kurioje yra daiktų interneto sistema, gali būti kaip komponentas aprašytas laiko automatu.

Paprastas daiktų interneto pavyzdys galėtų būti kambaryje esantys trys temperatūros jutikliai, kurie matuoja kambario temperatūrą, oro kondicionierius, kuris šaldo, šildytuvas, kuris šildo. Visi prietaisai

sujungti ir valdomi debesų kompiuterijos pagalba. Sistema turi užtikrinti, kad kambario temperatūra turi būti pasiekta ties norima reikšme, pavyzdžiui 26 ^{o}C [16].

Pradžioje reikėtų pradėti nuo fizinės aplinkos modeliavimo. Tam aplinkos modelyje yra įvedami



7 pav. Fizines aplinkos modelio schema [16]

kambario ir lauko temperatūrų kintamieji. Šie kintamieji turi pradines reikšmes, kurias galima priskirti prieš modelio simuliavimą. Jeigu kambario temperatūra yra mažesnė negu lauko, tai kambario temperatūra didės per tam tikrą laiko intervalą tam tikra verte (temperatūros vertė ir laiko intervalas yra sistemos parametrai ar konstantos, kurių konkrečios reikšmės yra pasirenkamos prieš modeliavimą), tačiau maksimali temperatūra neviršys lauko temperatūros. Jeigu kambario pradinė temperatūra yra didesnė nei lauko, tai kambario temperatūra mažės per tam tikrą laiko intervalą tam tikra verte. Kitaip tariant, kambario temperatūra turi artėti ir tapti lygi lauko temperatūrai (su sąlyga jeigu kambario temperatūros nekeičia koks nors kitas sistemos komponentas).

Kambario temperatūra yra nuskaitoma jutiklių. Jeigu oro kondicionierius pradeda šaldyti, tam tikru laiko žingsniu kambario temperatūra krenta 1 ^{o}C. Jeigu šildytuvas pradeda šildyti, tai atitinkamai tam tikru laiko žingsniu kambario temperatūra kyla 1 ^{o}C.

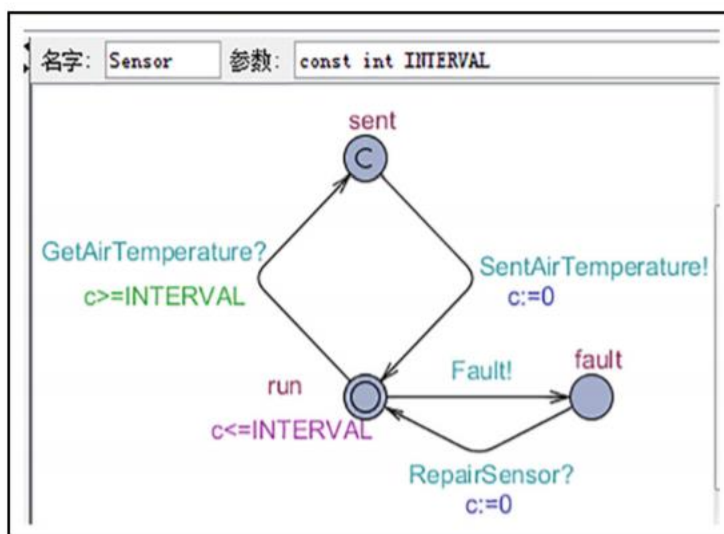
Ši sistema gali būti modeliuojama 1.1.4 skyriuje minėtu „*Uppaal*“ įrankiu. 7 pav. pavaizduotas fizinės aplinkos modelis. Konstanta *OUTSIDE_TEMPERATURE* žymima lauko temperatūra, *INIT_TEMPERATURE* žymima kambario temperatūra, *INTERVAL* – laiko intervalas, žingsnis tarp kiekvieno temperatūros pasikeitimo, INC žymimas temperatūros padidėjimas arba sumažėjimas, INCREMENT žymi temperatūros pokyčio vertė. 7 pav. pavaizduoti būsenos pasikeitimai

$$\langle loop, u\rangle \xrightarrow{\text{AirCool}} \langle loop, u'\rangle \text{ ir } \langle loop, u\rangle \xrightarrow{\text{AirHot}} \langle loop, u'\rangle$$

sinchronizuojasi su oro kondicionieriaus ir šildytuvo procesais atitinkamai kanalais $AirCool$ ir $AirHot$. Temperatūros nuskaitymo būsenos pasikeitimas

$$\langle loop, u\rangle \xrightarrow{\text{GetAirTemparature}} \langle loop, u'\rangle$$

sinchronizuojasi su jutiklio procesu kanalu $GetAirTemparature$. Šis būsenos pasikeitimas atitinka temperatūros jutiklio kambario temperatūros vertės nuskaitymą.



8 pav. Temperatūros jutiklio modelis [16]

Norint modeliuoti temperatūros jutiklį, reikalinga jutiklio atliekamų funkcijų analizė. Temperatūros jutiklis turi nuskaityti ir siuntinėti kambario temperatūros vertę per apibrėžtą laiko žingsnį. Modeliuojant šį įrenginį atsižvelgta, kad jis gali sugesti, todėl turi sugedimo (ang. *fault)* būseną. Temperatūros jutiklio modelio pavyzdys pateiktas 8 pav. Pavyzdyje pateiktas būsenos pasikeitimas:

$$\langle fault, u\rangle \xrightarrow{\text{RepairSensor?}} \langle run, u'\rangle.$$

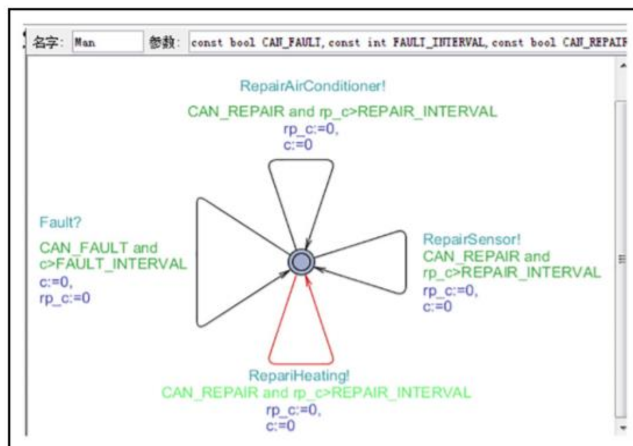
Šis pasikeitimas reiškia, kad po gedimo jutiklis gali būti pataisomas. Kanalu RepairSensor vyksta sinchronizacija su atskiru taisytojo procesu.

Kadangi šiame pavyzdyje nagrinėjama sistema turi trys identiškus komponentus (temperatūros jutiklius), galime pasinaudoti „*Uppaal*“ galimybe paskelbti kelis komponentus (procesus) sukurtus pagal tą patį vieną kartą aprašytą modelį (šabloną). 9 pav. matoma, kad deklaruojami trys temperatūros jutikliai *Sensor1*, *Sensor2*, *Sensor3* pagal modelį (šabloną) *Sensor*. Čia šablono parametro reikšmė 10 – laiko žingsnis, per kurį nuskaitoma ir siunčiama temperatūros vertė.

```
//Physical Environment in Summer and Winter
SummarAir=Air(50,35,100,true,1);    // Outdoor 50°C, indoor 35°C, increases by 1°C per 100 time units
WinterAir=Air(0,15,100,false,1);    // Outdoor 0°C, indoor 15°C, decreases by 1°C per 100 time units
// Temperature Sensors
Sensor1=Sensor(10);    // Temperature sensing and sending every 10 time units
Sensor2=Sensor(10);
Sensor3=Sensor(10);
// Air conditioning
AirCon1=AirCon(20);       // Temperature drops by 1°C per 20 time units under refrigeration
//Heating
Heating1=Heating(20);       // Temperature rises by 1°C per 20 time units under heating
//Cloud control service
HouseManager=Manager(5,10);    // Check whether the air conditioner or heating is fail every 5 time units
                               // Check whether all sensors are fail every 10 time units
// Artificial Repair
HouseMan=Man(true,0,false,0);    //All resource can fail at any time and cannot be repaired artificially
```

9 pav. Daiktų interneto sistemos deklaracija [16]

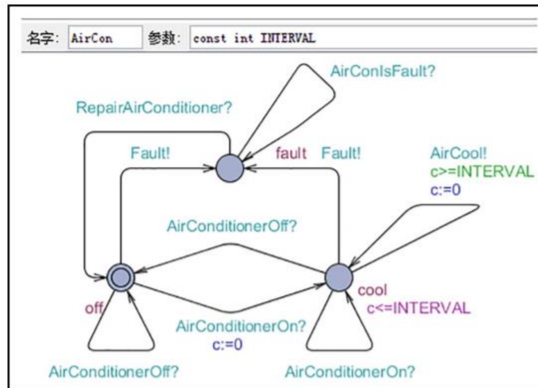
Tam, kad būtų apdorojami daiktų interneto įrenginių gedimai, modeliuojamas gedimų taisytojas (10 pav.). Sugęsti gali jutikliai, oro kondicionierius ir šildytuvas. Taisytojo modelyje apibrėžiami kintamieji, kurie nusako, ar daiktų interneto įrenginys gali sugesti, ar įrenginys gali būti pataisytas, per kokį laiko intervalą įrenginys gali būti pataisytas.



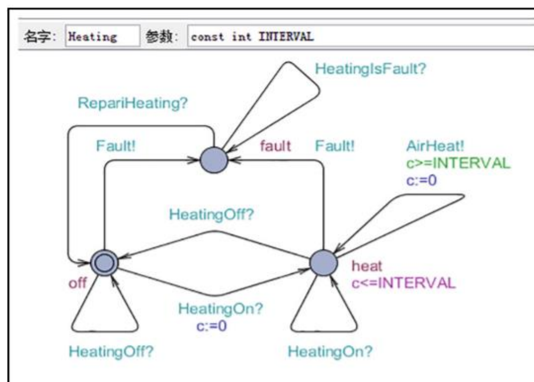
10 pav. Gedimų taisytojo modelis [16]

Oro kondicionieriaus modelis pavaizduotas 11 pav. Kai įrenginys gauna įjungimo pranešimą (įvyksta sinchronizacija su debesų kompiuterijos valdikliu per kanalą *AirConditionerOn)*, oro kondicionierius patenka į įjungimo būseną ir pradeda šaldyti (vyksta sinchronizacija su aplinkos procesu per kanalą *AirCool*), o kai gauna išjungimo pranešimą (įvyksta sinchronizacija su debesų

kompiuterijos valdikliu per kanalą *AirConditionerOff*) – patenka į išjungimo būseną ir nebešaldo. Jeigu oro kondicionierius yra veikiantis, jis visada turi sinchronizuotis kanalais *AirConditionerO\n Ir*



11 pav. Oro kondicionieriaus modelis [16]



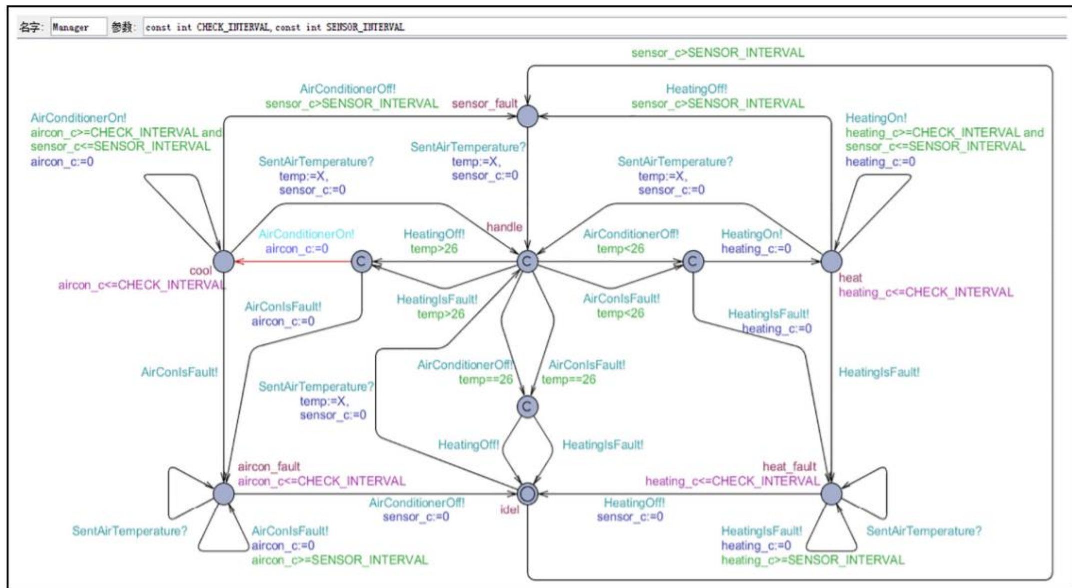
12 pav. Šildytuvo modelis [16]

AirConditionerOf . Jeigu įrenginys yra sugedęs, oro kondicionierius lieka sugedimo būsenoje *(fault)* kol taisytojas jo nepataiso. Šildytuvo modelis (12 pav.) yra analogiškas oro kondicionieriaus modeliui, skiriasi tik tuo, kad šis įrenginys šildo, o t.y. didina fizines aplinkos temperatūrą.

Daiktų interneto įrenginių valdiklis taip pat yra modeliuojamas. Jo modelis pavaizduotas 13 pav. Valdikio trys svarbiausios būsenos: *Idle*, kuomet nėra siunčiami pranešimai įrenginiams (kambario tempertūros vertė yra lygi norimai, nereikia atlikti šildymo arba šaldymo), *cool*, kuomet siunčiami pranešimai oro kondicionieriui mažinti aplinkos temperatūrai, *heat*, kuomet siunčiami pranešimai šildytuvui didinti aplinkos temperatūrai. Taip pat yra įdėti apribojimai ir papildomos būsenos, į kurias patenka valdiklis, kai laiko įrenginius sugedusiais. Pavyzdžiui, jeigu per nustatyta laiką negauna pranešimo iš temperatūros jutiklio, pereina į būsena *sensor_fault*.

Kai sistemos modelis sėkmingai aprašytas, jį galima verifikuoti, pavyzdžiui, patikrinti, ar sistema nesuges, ar neįvyks užstingimas. Užklausų pateikimo pavyzdys pavaizduotas 14 pav. [16].

Verifikavimas patvirtina, kad ši sistema parodo toleravimą gedimams, kuomet vienas jutiklis arba du jutikliai sugenda, sistema nesustoja veikti, taisytojas pataiso įrenginius ir sistema veikia toliau.



13 pav. Daiktų interneto įrenginių kontrolės taškas debesų kompiuterijoje [16]

```
Status
Established direct connection to local server.
(Academic) UPPAAL version 4.0.14 (rev. 5615), May 2014 -- server.
E<> Sensor1.fault
Property is satisfied.
E<> AirCon1.fault
Property is satisfied.
E<> Heating1.fault
Property is satisfied.
A[] not deadlock
Property is satisfied.
```

14 pav. IoT sistemos modelio verifikavimas [16]

A[] not deadlock tikrinama, ar sistema negali užstingti. *E<> Sensor1.fault*, *E<> AirCon1.fault*, *E<> Heating1.fault* tikrinama, ar įrenginiai gali sugesti.

Taigi, net jeigu daiktų interneto sistema nėra didelė, reikalingas tikslus kiekvieno sistemos komponento aprašymas. Modeliuojant didesnes ir įvairesnių komponentų bei fizikinių dydžių turinčias sistemas, atitinkamai atsirastų daugiau modelio procesų, kintamųjų ir lokacijų, lokacijų perėjimų.

2. Praktinė dalis

2.1 Modeliavimas

Šiame darbe buvo paimta daiktų interneto sistema iš [16] straipsnio, aprašyta šaltinių analizės 2.2.2 skyriuje, o praktinėje dalyje jos modelis išplėstas ir pakeistas taip, jog būtų galima atlikti formalų verifikavimą statistinių modelio patikrinimo būdu. [16] straipsnyje nebuvo pateiktas modelio išeities kodas (t.y. konkretus xml formato šablonas ir kintamųjų deklaracijos), todėl modeliuojant buvo remiamasi straipsnyje išdėstytais ir aprašytais algoritmais ir paveikslėliais. Tiriamą daiktų interneto sistemą sudaro tokie komponentai (procesai): aplinka - „*Environment*“, temperatūros jutiklis - „*TemperatureSensor*“, taisytojas - „*Repair*“, oro kondicionierius - „*AirConditioner*“, šildytuvas - „*Heating*“, debesų kompiuterijos valdiklis - „*CloudControl*“. Sistemos modelio deklaracija yra pavaizduota 15 pav.

Deklaracijoje nurodyti minėti procesai ir jų parametrų reikšmės. Aplinkos komponentas yra sukuriamas pagal parametrizuotą šabloną *Environment(50, 35, 100, true, 1)*, kur *50* yra lauko temperatūra, *35* yra pradinė kambario temperatūra, *100* - laiko žingsnis, po kurio lauko aplinka šildo arba šaldo kambarį nustatytu temperatūros pokyčiu, true – nustatoma, kad lauko aplinka šildo kambarį, *1* – temperatūros pokytis, kuriuo lauko aplinka šildo arba šaldo kambario aplinką.

Trys temperatūros sensoriai yra sukurti naudojant šabloną *TemperatureSensor(10)*, kur *10* – laiko intervalas, kas kiek laiko temperatūros jutiklis nuskaito ir nusiunčia temperatūrą debesų kompiuterijos valdikliui. Oro kondicionieriaus procesas yra sukuriamas su *AirConditioner(20)*, kur *20* – laiko intervalas, per kurį atšaldoma kambario aplinka vienu temperatūros vienetu. Šildytuvo procesas yra sukuriamas su *Heating(20)*, kur *20* – laiko intervalas, per kurį sušildoma kambario aplinka vienu temperatūros vienetu.

Debesų kompiuterijos valdiklis yra sukuriamas su *CloudControl(10, 20)*, kur *10* yra laiko reikšmė, pagal kurią debesų kompiuterijos valdiklis tikrina ar temperatūra buvo atsiųsta (temperatūros jutiklio gedimo tikrinimui), o *20* - laiko reikšmė, pagal kurią debesų kompiuterijos valdiklis tikrina, ar kambario temperatūra padidėjo arba sumažėjo (šildymo, šaldymo įrenginių gedimo tikrinimui). Pagaliau, taisytojo procesas yra sukuriamas naudojant šabloną *Repair(10)*, kur *10* yra laiko trukmė per kurią taisytojas patikrina ir pataiso įrenginį.

```
// Place template instantiations here.
EnvironmentProcess = Environment(50,35, 100,true, 1);

TemperatureSensor1 = TemperatureSensor(10);
TemperatureSensor2 = TemperatureSensor(10);
TemperatureSensor3 = TemperatureSensor(10);

AirConditionerProcess = AirConditioner(20);
HeatingProcess = Heating(20);

CloudControlProcess = CloudControl(10, 20);

RepairProcess = Repair(10);
// List one or more processes to be composed into a system.
system HeatingProcess, EnvironmentProcess, AirConditionerProcess, CloudControlProcess, TemperatureSensor1, TemperatureSensor2, TemperatureSensor3, RepairProcess;
```

15 pav. Sistemos deklaracija (atnaujinti)

Visi sistemos kanalai yra deklaruojami kaip transliavimo (ang. *broadcast channel*), kadangi to reikalauja „*Uppaal*“ įrankio SMC papildymas. Nuo paprasto kanalo transliuojamas kanalas skiriasi tuo, jog vienas procesas transliuoja, o daugiau nei vienas procesas klausosi to paties kanalo ir vykdo perėjimą tarp lokacijų, kai sulaukia transliavimo. Kad neįvyktų atsitiktinių perėjimų į lokacijas, kai kuriuose sistemos perėjimuose reikėjo pridėti papildomų perėjimo sąlygų, kad procesai veiktų tinkamai bei, kad būtų įmanoma procesų sinchronizacija.

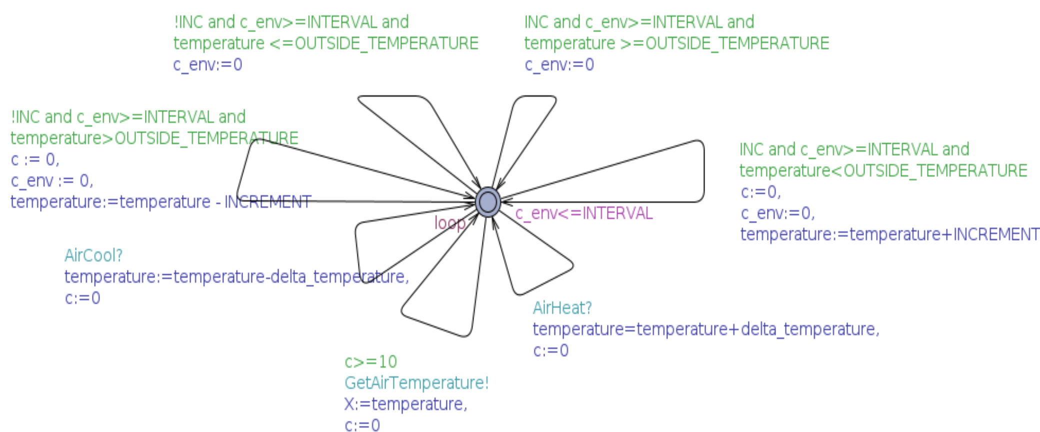
Globalios deklaracijos pavaizduotos 16 pav. Šioje vietoje skelbiami globalūs kintamieji reikalingi daugiau nei vienam procesui. Pavyzdžiui, deklaruojama temperatūra, kurią norima palaikyti (kintamasis *target_temperature = 26*) ir jutiklio nuskaityta temperatūra (kintamasis *X*), kurią naudoja jutiklio ir debesų kompiuterijos valdiklio procesai. „*delta_temperature*“ konstanta nustato kokiu laipsniu šildoma arba šaldoma aplinka (naudoja šildytuvo ir oro kondicionieriaus procesai). Taip pat tarp globalių deklaracijų yra pervadinti kanalai, kurie siejasi su įrenginių gedimais: jutiklio gedimo - „*SFault*”, oro kondicionieriaus gedimo - „*CFault*”, šildytuvo - „*HFault*“. Šie kanalai buvo atskirti dėl to, jog taisytojui būtų transliuojama, kuris įrenginys sugedo. Kitų kanalų pavadinimai nebuvo pakeisti. Detalesni modelio procesų aprašymai yra kituose skyriuose.

```
//target temperature
int target_temperature = 26;

 //sensor sensed temperature that is later sent to cloud
int X;

//how much degrees does cooling / heating device change the tempeture
const int delta_temperature = 1;

//channels
broadcast chan AirCool, AirHeat;
broadcast chan GetAirTemperature, SentAirTemperature;
broadcast chan SFault, CFault, HFault;
broadcast chan RepairSensor, RepairAirConditioner, RepairHeating;
broadcast chan AirConditionerOn, AirConditionerOff, HeatingOn, HeatingOff;

clock t;
```

16 pav. Globaliųjų kintamųjų deklaracija

2.1.1 Aplinkos procesas

Aplinkos procesas yra modifikuotas tuo, jog buvo pridėta apsauga (papildoma sąlyga) prie perėjimo, kuris turi transliavimo kanalą GetAirTemperature!, susietą su jutiklio modeliu. Aplinkos procesas pavaizduotas 17 pav. Perėjimas, kuriame yra pridėtas apribojimas *c >= 10,* leidžia aplinkai transliuoti temperatūrą kas 10 laiko vienetų. Jeigu šios perėjimo sąlygos nebūtų, verifikuojant grąžinama klaida, kad sistemą per ribotą laiką atlieką begalybe veiksmų („Probably zeno behavior“), taip yra todėl, kad aplinka stochastiškai per dažnai atlieka temperatūros transliavimą.



17 pav. Aplinkos procesas

2.1.2 Temperatūros jutiklio procesas

Temperatūros jutikliui reikia nuskaityti temperatūrą iš aplinkos proceso. Paleidus jutiklio ir aplinkos simuliacijas buvo pastebėta, kad verifikuojant SMC būdu negalima toje pačioje viršūnėje konfigūruoti apribojimo (angl *guard*) ir klausymosi kanalo *GetAirTemperature?* Kai kanalai modelyje buvo pakeisti iš paprastų į transliavimo, buvo pastebėta, kad jutiklis staigiai pereina į gedimo būseną. Dėl šios priežasties buvo naudojamos viršūnės su tikimybiniais svoriais ir pridėta papildoma lokacija „*sense*“. Iš viso procesą sudaro lokacijos *sleep*, *sense*, *sent* ir *fault*. Lokacijoje *sleep* išlieka invariantas

c <= INTERVAL

čia *c* – lokaliai deklaruotas laikrodis, *INTERVAL* – laiko tarpas per kurį temperatūros jutiklis nuskaito ir išsiunčia temperatūrą. Iš lokacijos *sleep* einantys perėjimai:

- Perėjimas, kuris eina į lokaciją *sense* su nustatytu tikimybiniu svoriu arba į lokaciją *fault* su nustatytu tikimybiniu svoriu. Perėjimas turi *c >= INTERVAL* perėjimo sąlygą. Kai perėjimas

vyksta į *sense* lokaciją, tai laikoma, kad įrenginys atliks temperatūros nuskaitymą, o kai perėjimas vyksta į *fault* lokaciją, tai laikoma, kad įrenginyje įvyko gedimas.

Iš lokacijos *sense* einantys perėjimai:

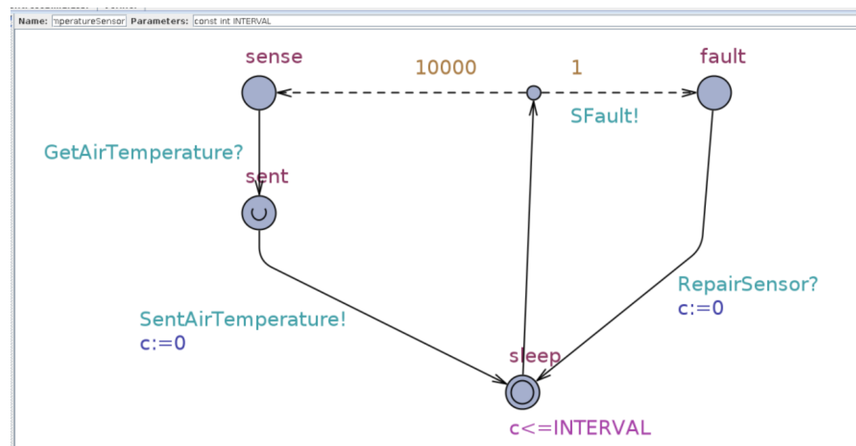
- Perėjimas, kuris eina į lokaciją *sent* per *GetAirTemperature?* kanalą. Perėjimas vyksta tuomet, kai temperatūros jutiklis nuskaito temperatūrą iš aplinkos (aplinka transliuoja kanalu *GetAirTemperature!*). [16] straipsnyje temperatūros nuskaitymui ir išsiuntimui buvo naudojama viena lokacija, tačiau tokiu atveju leidžiant simuliacija buvo pastebėta, kad simuliacijos pradžioje kartais išsiunčiama temperatūros pradinė vertė 0, kuri neatitinka aplinkos transliuojamos temperatūros, kurios vertė buvo 35. Kai buvo atskirta į dvi lokacijas, ši situacija nebeatsikartojo.

Lokacijos *sent* tipas yra skubus (ang. *Urgent*) ir pridėto laiko aprašyto invarianto, o tai reiškia, kad perėjus į šią lokaciją procesas nedelsiant pereina į *sleep* lokaciją. Perėjimas į *sleep* lokaciją kanalu *SentAirTempeture!* Reiškia, kad temperatūra išsiunčiama debesų kompiuterijos valdikliui.

Iš lokacijos *fault* einantys perėjimai:

- Perėjimas, kuris eina į lokaciją *sleep* per *RepairSensor?* kanalą. Perėjimas vyksta tuomet, kai taisytojas pataiso temperatūros jutiklį.

Minėtose perėjimuose lokaliai deklaruoto laikrodžio vertė *c* visuomet grąžinama į *0* vertę. Modelis pavaizduotas 18 pav. Paveikslėlyje pavaizduoto proceso gedimo tikimybinis svoris yra 1, o veikimo 10000. Kitais žodžiais, su tikimybe 1/10001 procesas pereis į būseną (lokaciją) *fault*, o su tikimybe 10000/10001 į lokaciją *sense*.

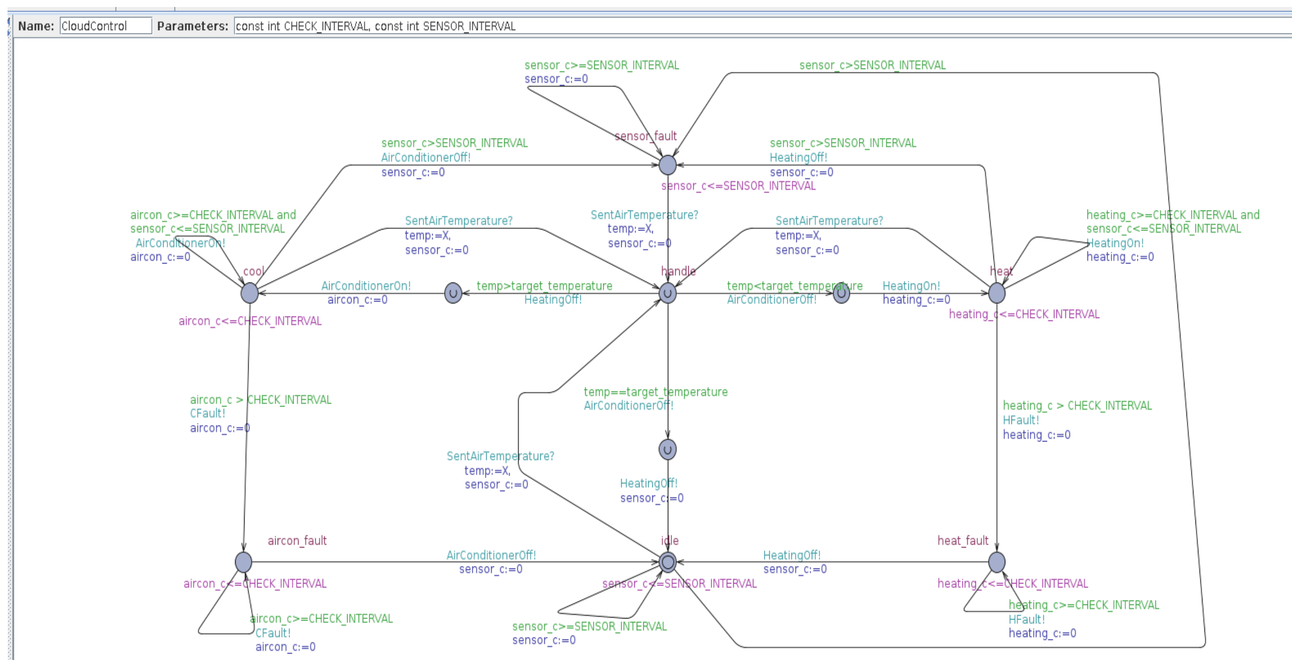


18 pav. Temperatūros jutiklio procesas

2.1.3 Debesų kompiuterijos valdiklio procesas

Debesų kompiuterijos valdiklio procesas pavaizduotas 19 pav. Procesas susideda iš lokacijų *idle*, *handle*, *sensor_fault*, *cool*, *heat*, *aircon_fault*, *heating_fault* ir kelių skubių (ang. *Urgent*) lokacijų. Simuliuojant pastebėta, kad sistemos veikimui įtakos nedaro, jei įsipareigojusios (ang. *commited*) lokacijos pakeičiamos į skubias. Skubios lokacijos čia yra todėl, kad nedelsiant vienu metu būtų išjungiamas šildytuvas ir oro kondicionierius (kai kambario temperatūra lygi *target_temperature*) bei vienu metu išjungiamas šildytuvas ir įjungiamas oro kondicionierius ir atvirkščiai (kai kambario temperatūra nelygi *target_temperature*). Taip pat deklaruoti lokalūs laikrodžiai:

- *sensor_c*, kuris skaičiuoja laiką nuo paskutinio temperatūros jutiklio veiksmo;
- *heating_c*, kuris skaičiuoja laiką nuo paskutinio šildytuvo veiksmo;
- *aircon_c*, kuris skaičiuoja laiką nuo paskutinio oro kondicionieriaus veiksmo.



19 pav. Debesų kompiuterijos valdiklio procesas

Modelis buvo pakeistas taip, kad nebūtų užstingimo laike, o t.y. kiekvienoje neskubioje lokacijoje pasirūpinta, jog būtų laiko invariantai

sensor_c <= SENSOR_INTERVAL arba *heating_c <= CHECK_INTERVAL* arba

aircon_c <= CHECK_INTERVAL

Čia *SENSOR_INTERVAL* – laiko tarpas per kurį temperatūros jutiklis turėtų atsiųsti temperatūros. Jei šis laiko tarpas viršijamas, debesų kompiuterijos valdiklis laiko jutiklį sugedusiu.. *CHECK_INTERVAL* – laiko tarpas per kurį oro kondicionierius turėtų šaldyti arba šildytuvas šildyti. Jei šis laiko tarpas viršijamas, debesų kompiuterijos valdiklis laiko įrenginius sugedusiais.

Debesų kompiuterijos valdiklio pradinė lokacija yra *idle*. Iš lokacijos *idle* einantys perėjimai:

- Perėjimas, kuris eina į lokaciją *handle* per *SentAirTemperature?* kanalą. Perėjimas vyksta tuomet, kai temperatūros jutiklis atsiunčia patalpos temperatūros vertę į debesų kompiuterijos valdiklį. Kintamasis *temp* yra lokalusis kintamasis skirtas gautai temperatūrai saugoti.

- Perėjimas, kuris eina į tą pačią lokaciją su *sensor_c >= SENSOR_INTERVAL* perėjimo sąlyga. Šis perėjimas buvo pridėtas, kad nebūtų užstingimo laike. Šis perėjimas reiškia, debesų kompiuterijos valdiklis lieka laisvoje būsenoje ir neatlieka apdorojimo.

- Perėjimas, kuris eina į lokaciją *sensor_fault* su *sensor_c > SENSOR_INTERVAL sąlyga*. Perėjimas vyksta tuomet, kai temperatūros jutiklis neatsiunčia patalpos temperatūros vertės į debesų kompiuterijos valdiklį per laiką *SENSOR_INTERVAL*.

Iš lokacijos *sensor_fault* perėjimai:

- Perėjimas, kuris eina į lokaciją *handle* per *SentAirTemperature?* kanalą. Perėjimas vyksta tuomet, kai temperatūros jutiklis atsiunčia patalpos temperatūros vertę į debesų kompiuterijos valdiklį. Kintamasis *temp* yra lokalusis kintamasis skirtas gautai temperatūrai saugoti.

- Perėjimas, kuris eina į tą pačią lokaciją su *sensor_c >= SENSOR_INTERVAL* perėjimo sąlyga. Šis perėjimas buvo pridėtas, kad nebūtų užstingimo laike. Šis perėjimas reiškia, kad debesų kompiuterijos valdiklis temperatūros jutiklį laiko sugedusiu.

Iš lokacijos *handle* perėjimai:

- Perėjimas, kuris eina per skubią (angl *Urgent*) lokaciją su sąlyga *temp > target_temperature* kanalu *HeatingOff!* į lokaciją *cool* kanalu *AirConditionerOn!*. Perėjimas vyksta tuomet, kai temperatūros jutiklis atsiunčia patalpos temperatūros vertę, kuri yra didesnė už *target_temperature*, į debesų kompiuterijos valdiklį. Valdiklis nusprendžia, jog reikalingas šaldymas, todėl įjungiamas oro kondicionierius ir išjungiamas šildytuvas.

- Perėjimas, kuris eina per skubią (angl *Urgent*) lokaciją su sąlyga *temp < target_temperature* kanalu *AirConditionerOff!* į lokaciją *heat* kanalu *HeatingOn!*. Perėjimas vyksta tuomet, kai temperatūros jutiklis atsiunčia patalpos temperatūros vertę, kuri yra mažesnė už *target_temperature*, į debesų kompiuterijos valdiklį. Valdiklis nusprendžia, jog reikalingas šildymas, todėl įjungiamas šildytuvas ir išjungiamas oro kondicionierius.

- Perėjimas, kuris eina per skubią (angl *Urgent*) lokaciją su sąlyga *temp == target_temperature* kanalu *AirConditionerOff!* į lokaciją *idle* kanalu *HeatingOff!*. Perėjimas vyksta tuomet, kai temperatūros jutiklis atsiunčia patalpos temperatūros vertę, kuri yra lygi *target_temperature* vertei, į debesų kompiuterijos valdiklį. Valdiklis nusprendžia, jog nereikalingas nei šildymas, nei šaldymas, todėl užtikrinama, jog šildytuvas oro kondicionierius išjungti.

Iš lokacijos *cool* einantys perėjimai:

- Perėjimas, kuris eina į lokaciją *sensor_fault* transliuojant *AirConditionerOff?* kanalu su sąlyga *sensor_c > SENSOR_INTERVAL*. Perėjimas vyksta tuomet, kai temperatūros jutiklis neatsiunčia patalpos temperatūros vertės į debesų kompiuterijos valdiklį per laiką *SENSOR_INTERVAL*. Šiame perėjime pasirūpinama, kad oro kondicionierius išsijungtų.

- Perėjimas, kuris eina į lokaciją *handle* per *SentAirTemperature?* kanalą. Perėjimas vyksta tuomet, kai temperatūros jutiklis atsiunčia patalpos temperatūros vertę į debesų kompiuterijos valdiklį.

- Perėjimas, kuris eina į tą pačią lokaciją su sąlygomis *sensor_c <= SENSOR_INTERVAL ir aircon_c >= CHECK_INTERVAL* transliuojant *AirConditionerOn!* Kanalu. Šis perėjimas reiškia, jog debesų kompiuterijos valdiklis rūpinasi, kad oro kondicionierius liktų įjungtas.

- Perėjimas, kuris eina į lokaciją *aircon_fault* su sąlyga *aircon_c > CHECK_INTERVAL* transliuojant *CFault!* kanalu. Šis perėjimas reiškia, jog debesų kompiuterijos valdiklis nustato, jog oro kondicionierius sugedo.

Iš lokacijos *heat* einantys perėjimai:

- Perėjimas, kuris eina į lokaciją *sensor_fault* transliuojant *HeatingOff?* kanalu su sąlyga *sensor_c > SENSOR_INTERVAL*. Perėjimas vyksta tuomet, kai temperatūros jutiklis neatsiunčia patalpos temperatūros vertės į debesų kompiuterijos valdiklį per laiką *SENSOR_INTERVAL*. Šiame perėjime pasirūpinama, kad šildytuvas išsijungtų.

- Perėjimas, kuris eina į lokaciją *handle* per *SentAirTemperature?* kanalą. Perėjimas vyksta tuomet, kai temperatūros jutiklis atsiunčia patalpos temperatūros vertę į debesų kompiuterijos valdiklį.
- Perėjimas, kuris eina į tą pačią lokaciją su sąlygomis *sensor_c <= SENSOR_INTERVAL ir heating_c >= CHECK_INTERVAL* transliuojant *HeatingOn!* Kanalu. Šis perėjimas reiškia, jog debesų kompiuterijos valdiklis rūpinasi, kad šildytuvas liktų įjungtas.
- Perėjimas, kuris eina į lokaciją *heating_fault* su sąlyga *heating_c > CHECK_INTERVAL* transliuojant *HFault!* kanalu. Šis perėjimas reiškia, jog debesų kompiuterijos valdiklis nustato, jog šildytuvas sugedo.

Iš lokacijos *aircon_fault* einantys perėjimai:

- Perėjimas, kuris eina į tą pačią lokaciją su *aircon_c >= CHECK_INTERVAL* perėjimo sąlyga. Šis perėjimas buvo pridėtas, kad nebūtų užstingimo laike. Šis perėjimas reiškia, kad debesų kompiuterijos valdiklis oro kondicionierių laiko sugedusiu.
- Perėjimas, kuris eina į lokaciją *idle*. Šis perėjimas reiškia, kad debesų kompiuterijos valdiklio paruošiamas priimti kitą temperatūros vertę.

Iš lokacijos *heating_fault* einantys perėjimai:

- Perėjimas, kuris eina į tą pačią lokaciją su *heating_c >= CHECK_INTERVAL* perėjimo sąlyga. Šis perėjimas buvo pridėtas, kad nebūtų užstingimo laike. Šis perėjimas reiškia, kad debesų kompiuterijos valdiklis šildytuvą laiko sugedusiu.
- Perėjimas, kuris eina į lokaciją *idle*. Šis perėjimas reiškia, kad debesų kompiuterijos valdiklis pasiruošęs priimti kitą temperatūros vertę.

2.1.4 Oro kondicionieriaus procesas

Oro kondicionieriaus procesas buvo pakeistas taip, kad nebūtų užstingimo laike, o t.y. kiekvienoje lokacijoje pridėtas invariantas

c <= INTERVAL

čia c – lokaliai deklaruotas laikrodis, *INTERVAL* – laiko tarpas per kurį įrenginys šaldo patalpą. Procesą sudaro lokacijos *off*, *cool* ir *fault*. Iš lokacijos *off* einantys perėjimai:

- Perėjimas, kuris eina į lokaciją *cool* per *AirConditionerOn?* kanalą. Perėjimas vyksta tuomet, kai debesų kompiuterijos valdiklis siunčia signalą šaldyti patalpą.

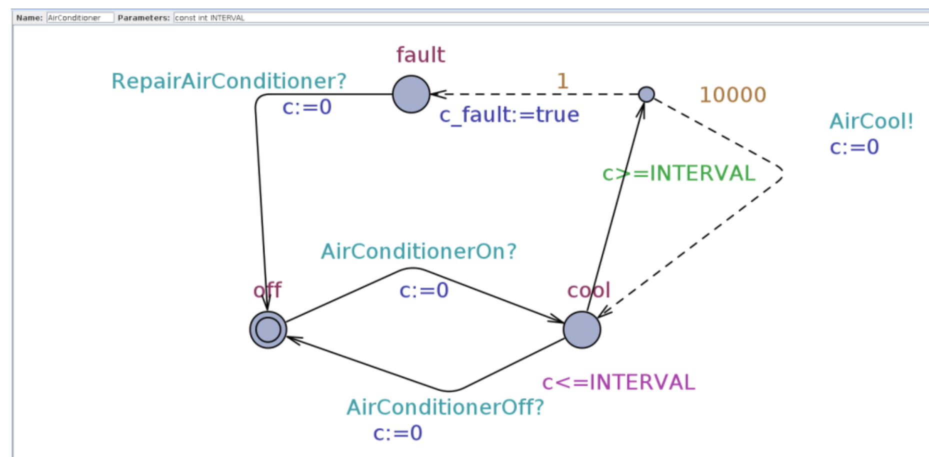
Iš lokacijos *cool* einantys perėjimai:

- Perėjimas, kuris eina į lokaciją *off* per *AirConditionerOff?* kanalą. Perėjimas vyksta tuomet, kai debesų kompiuterijos valdiklis siunčia išjungimo signalą.
- Perėjimas, kuris eina į tą pačią lokaciją transliuojant kanalu *AirCool!* su nustatytu tikimybiniu svoriu arba į lokaciją fault su nustatytu tikimybiniu svoriu. Perėjimas turi $c >= INTERVAL$ perėjimo sąlygą. Kai perėjimas vyksta į tą pačią lokaciją, tai laikoma, kad įrenginys lieka įjungtas, o kai perėjimas vyksta į fault lokaciją, tai laikoma, kad įrenginys sugenda. Tikimybiniai svoriai parenkami taip, kad sugedimas yra mažiau tikėtinas nei šaldymas.

Iš lokacijos *fault* einantys perėjimai:

- Perėjimas, kuris eina į lokaciją *off* per *RepairAirConditioner?* kanalą. Perėjimas vyksta tuomet, kai taisytojas pataiso oro kondicionierių.

Minėtose perėjimuose lokaliai deklaruoto laikrodžio vertė *c* visuomet grąžinama į 0 vertę. Procesas pavaizduotas 20 pav.



20 pav. Oro kondicionieriaus procesas

2.1.5 Šildytuvo procesas

Šildytuvo procesas buvo pakeistas taip, kad nebūtų užstingimo laike, o t.y. kiekvienoje lokacijoje pridėtas invariantas

c <= INTERVAL

čia c – lokaliai deklaruotas laikrodis, *INTERVAL* – laiko tarpas per kurį įrenginys šildo patalpą. Modelį sudaro lokacijos *off*, *heat* ir *fault*. Iš lokacijos *off* einantys perėjimai:

- Perėjimas, kuris eina į lokaciją *heat* per *HeatingOn?* kanalą. Perėjimas vyksta tuomet, kai debesų kompiuterijos valdiklis siunčia signalą šildyti patalpą.

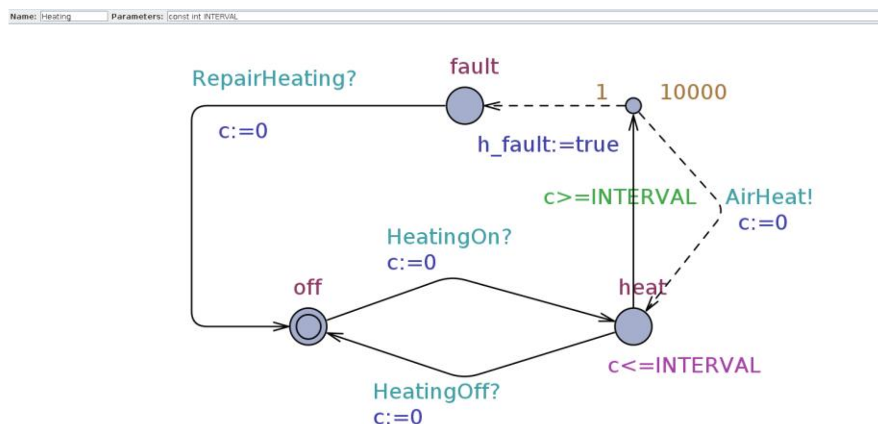
Iš lokacijos *heat* einantys perėjimai:

- Perėjimas, kuris eina į lokaciją *off* per *HeatingOff?* kanalą. Perėjimas vyksta tuomet, kai debesų kompiuterijos valdiklis siunčia išjungimo signalą.
- Perėjimas, kuris eina į tą pačią lokacija transliuojant kanalu *AirHeat!* su nustatytu tikimybiniu svoriu arba į lokaciją *fault* su nustatytu tikimybiniu svoriu. Perėjimas turi *c >= INTERVAL* perėjimo sąlygą. Kai perėjimas vyksta į tą pačią lokaciją, tai laikoma, kad įrenginys lieka įjungtas, o kai perėjimas vyksta į *fault* lokaciją, tai laikoma, kad įrenginys sugenda. Tikimybiniai svoriai parenkami taip, kad sugedimas yra mažiau tikėtinas nei šildymas.

Iš lokacijos *fault* einantys perėjimai:

- Perėjimas, kuris eina į lokaciją *off* per *RepairHeating?* kanalą. Perėjimas vyksta tuomet, kai taisytojas pataiso šildytuvą.

Minėtose perėjimuose lokaliai deklaruoto laikrodžio vertė *c* visuomet grąžinama į 0 vertę. Modelis pavaizduotas 21 pav.



21 pav. Šildytuvo procesas

2.1.6 Taisytojo procesas

Taisytojo procesas pasikeitė tuo, jog klausomasi jutiklio, oro kondicionieriaus, ir šildytuvo gedimų atitinkamai *SFault?*, *Hfault?* Ir *Cfault?* kanalais išvestose perėjimuose. Šiais kanalais tarsi atkeliauja pranešimas, kad įrenginys sugedo. Lokacija išlieka tik viena ir ją papildo invariantas:

rp_c <= REPAIR_INTERVAL

kur *rp_c* – lokaliai deklaruotas laikrodis, *REPAIR_INTERVAL* – laiko intervalas per kurį pataisomi įrenginiai. Lokaliai deklaruoti kintamieji *s_fault*, *h_fault*, *c_fault*, kuriais žymimos įrenginių gedimo būsenos. Kintamojo *s_fault* reikšmė tampa true, kai gedimas įvyksta temperatūros jutiklyje, *h_fault* tampa *true*, kai gedimas įvyksta šildytuve, *c_fault* tampa *true*, kai gedimas įvyksta oro kondicionieriuje. Be minėtų perėjimų iš lokacijos išvesti keturi perėjimai:

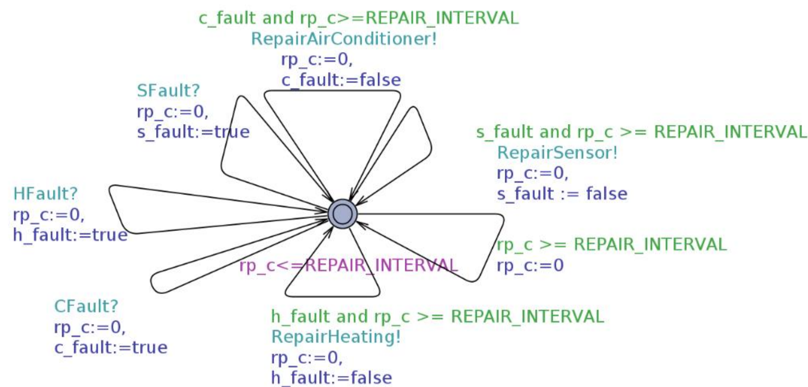
- Perėjimas, kurio sąlyga yra *c_fault and rp_c >= REPAIR_INTERVAL*, transliavimas kanalu *RepairAirConditioner!*. Po perėjimo kintamasis *c_fault* gauna *false* reikšmę. Šis perėjimas modeliuoja oro kondicionieriaus pataisymą.

- Perėjimas, kurio sąlyga yra *s_fault and rp_c >= REPAIR_INTERVAL*, transliavimas kanalu *RepairSensor!*. Po perėjimo kintamasis *s_fault* gauna *false* reikšmę. Šis perėjimas modeliuoja jutiklio pataisymą.

- Perėjimas, kurio sąlyga yra *h_fault and rp_c >= REPAIR_INTERVAL*, transliavimas kanalu *RepairHeater!*. Po perėjimo kintamasis *h_fault* gauna *false* reikšmę. Šis perėjimas modeliuoja oro kondicionieriaus pataisymą.

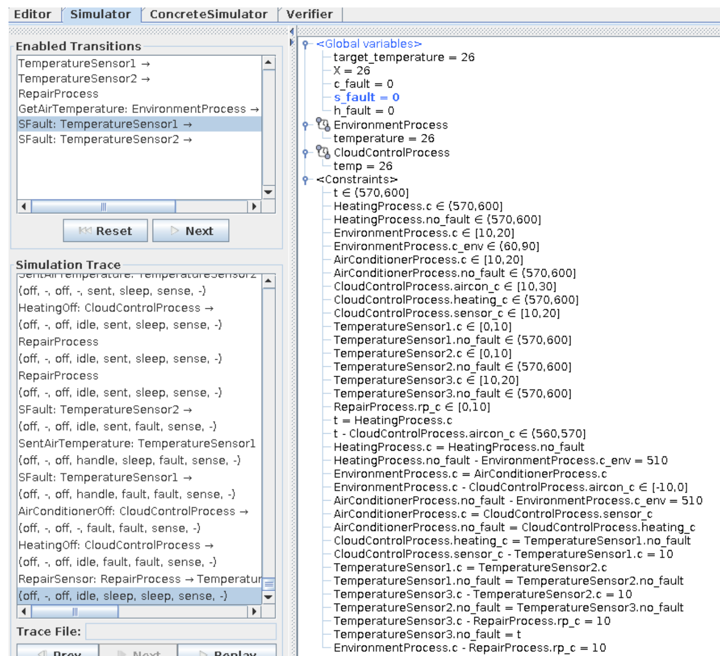
- Perėjimas, kurio sąlyga yra *rp_c >= REPAIR_INTERVAL*. Šis perėjimas reikalingas, kad neįvyktų laiko užstingimas arba galima laikyti, kad taisytojas nieko nedaro, kai nėra ką taisyti.

Visuose minėtuose perėjimuose lokalus laikrodis *rp_c* nustatomas vėl į pradinę reikšmę - 0. Taisytojo modelis „*Uppaal*“ įrankyje pavaizduotas 22 pav.



22 pav. Taisytojo procesas

Sumodeliavus sistemą ir paleidus jos simuliaciją „*Uppaal*“ įrankiu simuliatoriaus sekcijoje matoma, kaip modelis veikia, į kokias lokacijas pereina, kaip keičiasi kintamieji (žr. 22 pav.). Pagal minėtą deklaraciją 15 pav. patalpos temperatūra *temperature* mažėja. Nuo 35 temperatūros vienetų buvo atšaldyta iki 26 nepraėjus 2000 laiko vienetų.

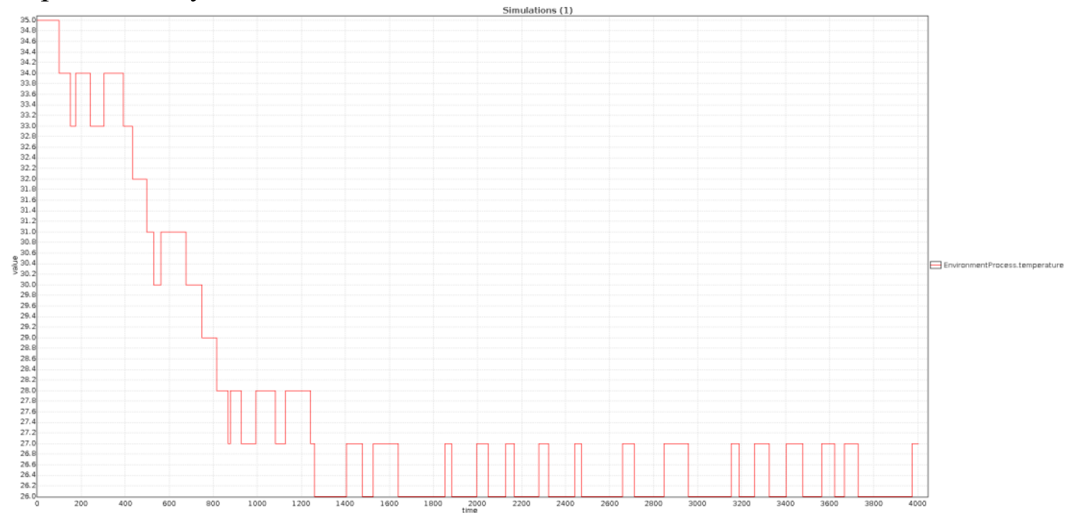


22 pav. Simuliatoriaus posistemė

Pavaizduoti kintamieji ir laikrodžiai, kurių vertės keičiasi, kuomet veikia simuliacija. Simuliacija taip pat galima stebėti įrankio verifikavimo skiltyje įvedus funkciją *simulate*. Pavyzdžiui, įvedus formulę:

simulate [<= 4000; 1] {Environment.temperature}

gaunamas rezultatas yra grafikas, kuriame pavaizduota, kaip kinta aplinkos temperatūra priklausomai nuo laiko, kai laiko vienetai yra nuo 0 iki 4000, o papildomas parametras *1* reiškia vieną sistemos simuliacijos paleidimą (ang. *run*). Rezultatas pavaizduotas 23 pav. Temperatūros vertės nuo 35 vienetų sumažėjamas iki 26 vienetų reiškia oro kondicionieriaus veikimą. Matomi šuoliai nuo 26 iki 27 yra dėl aplinkos šildymo.



23 pav. *simulate [<= 4000; 1] {Environment.temperature}* užklausos rezultatas

2.2 Verifikavimas

Šiame skyriuje pateikiamas 2.1 skyriuje aprašyto sistemos modelio verifikavimas klasikinio modelio patikrinimo bei statistinio modelio patikrinimo formulėmis. Sistemos parametrai nustatyti taip, kaip pavaizduota 15 pav. Pirmoji verifikuojama savybė buvo aklaviečių nebuvimo (ang. *deadlock freeness*). Ši savybė buvo verifikuota įvedus formule:

A[] not deadlock

Įvedus šią užklausą įrankio verifikavimo skiltyje buvo gauta, kad sistema aklaviečių neturi. Įrankis grąžino rezultatą „Property statisfied“.

Kita verifikuotina savybė yra funkcinis teisingumas. Patikrinama, ar sistemoje egzistuoja būsena, kurioje dalyvaujantys įrenginiai gali sugesti, formulėmis:

E<> TemperatureSensor1.fault

E<> AirConditionierProcess.fault

E<> HeatingProcess.fault

Sistemos veikimo metu negali egzistuoti būsena, kurioje oro kondicionierius ir šildytuvas bus vienu metu įjungti. Toks reikalavimas verifikuojamas formule:

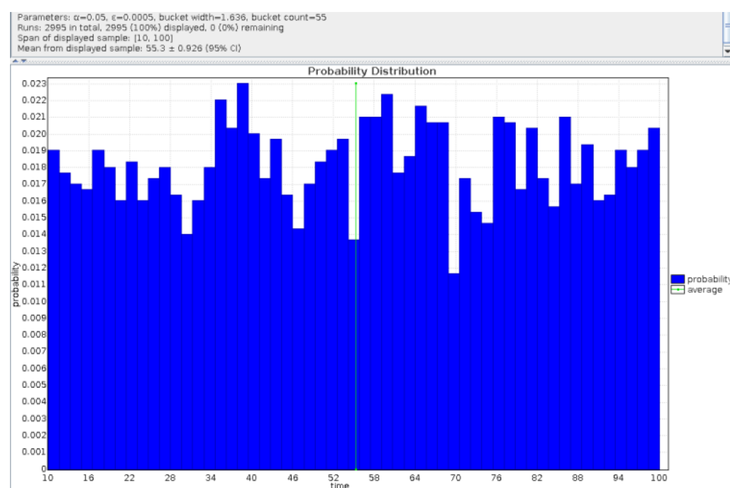
A[] not AirConditionierProcess.cool and HeatingProcess.heat

Minėtomis formulėmis sistema buvo verifikuota ir [16] straipsnyje, kuomet nebuvo atliktos modelio modifikacijos, leidžiančios verifikuoti SMC būdu. Po modelio modifikacijų šį verifikavimą svarbu pakartotinai atlikti norint įsitikinti, ar modifikacijos nepakeitė modelio taip, kad nebetenkintų minėtų savybių.

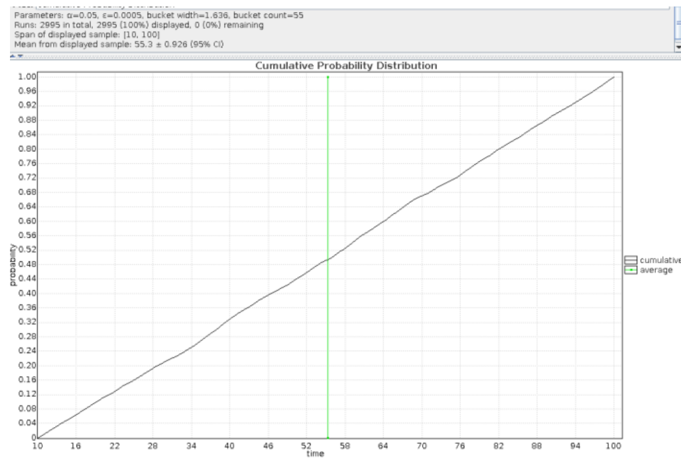
Pereinant prie verifikavimo SMC metodu, pirma tikrinama savybė buvo suformuluota tokia laiko logikos formule su laiko apribojimu:

Pr [t <= 5000] (<>AirConditionerProcess.cool)

Ši formulė užklausia tikimybinio skirstinio laiko intervale nuo 0 iki 5000 oro kondicionieriaus šaldymo būsenos (kitais žodžiais, kaip tikėtina, kad vykdymo metu oro kondicionieriaus bus lokacijoje „*cool*“).. Toks skirstinys pavaizduotas 24 pav. ir 25 pav. Parodo, kad tikimybė yra labai didelė (arti 1). Sistemos deklaracijoje patalpos temperatūra, kurios vertė yra *35*, yra didesnė už norimą *target_temperature*, kurios vertė yra *26*. Todėl kambarį reikia šaldyti ir pagal sistemos aprašymą turėtų įsijungti oro kondicionieriaus. Visų įrenginių gedimo tikimybinis svoris yra *1*, o veikimo *10000*. „*Uppaal*“ vienintelis statistikos parametras, kuris buvo pakeistas, yra tikimybės neapibrėžtumas (ang. *probability uncertainty*) ε. Parametro vertė nustatyta į $5 \cdot 10^{-4}$. Toks pakeitimas leido „*Uppaal*“ įrankiui grąžinti tikslesnius grafinius rezultatus (tikimybinius pasiskirstymus).

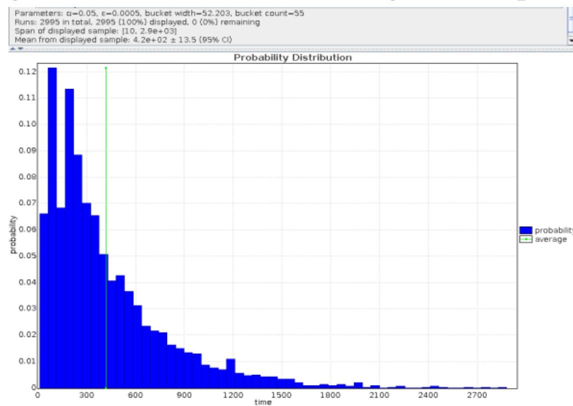


24 pav. Oro kondicionieriaus įjungimo būsenos tikimybinis skirstinys, kai iš 35 temperatūros vienetų reikalingas atšaldymas į 26.

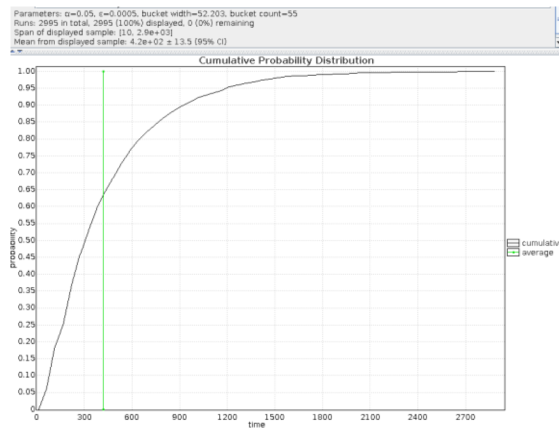


24 pav. Oro kondicionieriaus įjungimo būsenos pasiskirstymo funkcija, kai iš 35 temperatūros vertės reikalingas atšaldymas į 26.

Padidinus temperatūros jutiklio gedimo tikimybini svorį iš 1 į $1 \cdot 10^5$ tos pačios užklausos rezultatas pavaizduotas 25 pav. ir 26 pav. Tikimybė, kad oro kondicienirius bus įjungtas, yra pradžioje didžiausia. Kadangi jutiklio gedimo tikimybė didelė, jutikliai dažniau pradeda gesti ir debesų kompiuterijos valdiklis išjungia oro kondicionierių, kai negauna temperatūros pranešimo.



25 pav. Oro kondicionieriaus įjungimo būsenos tikimybės skirstinys, kai jutiklio gedimas yra daugiau tikėtinas, kai iš 35 temperatūros vertės reikalingas atšaldymas į 26.



26 pav. Oro kondicionieriaus įjungimo būsenos pasiskirstymo funkcija, kai jutiklio gedimas yra mažiau tikėtinas nei veikimas ir iš 35 temperatūros vertės reikalingas atšaldymas į 26.

Kita dominanti savybė, kuri tikrina, kad šildytuvas ilgainiui bus šildymo būsenoje, buvo suformuluota taip:

Pr [t <= 5000] (<>HeatingProcess.heat)

Rezultatas buvo „Property satisfied“, bet su maža tikimybę. Simuliacijoje buvo pastebėta, kad šiame scenarijuje iš tiesų gali būti taip, kad pasiekus norima temperatūra (26 vienetai) oro kondicionierius neiškart įsijungia ir dar vis dar šaldo patalpą, tuomet, kai temperatūra mažesnė nei 26, įjungiamas šildytuvas ir dėl to egzistuoja maža tikimybė, kad šildytuvas gali įsijungti.

Pakeitus pradinių parametrų reikšmes taip, kad patalpos temperatūra pradžioje yra ne 35, o 15 ir lauko – 10, buvo kartojamas užklausų pateikimas. Pateikus užklausą

Pr [t <= 5000] (<>AirConditionerProcess.cool)

Rezultatas buvo „Property satisfied“. Simuliacijoje buvo pastebėta, kad šiame scenarijuje (panašiai kaip ankstesniame) iš tiesų gali būti taip, kad pasiekus norima temperatūra (26 vienetai) šildytuvas ne iš karto išsijungia ir dar vis dar šildo patalpą, tuomet, kai temperatūra didesnė nei 26, įjungiamas oro kondicionierius ir dėl to egzistuoja maža tikimybė, kad oro kondicionierius gali būti įjungtas.

Pateikus užklausą

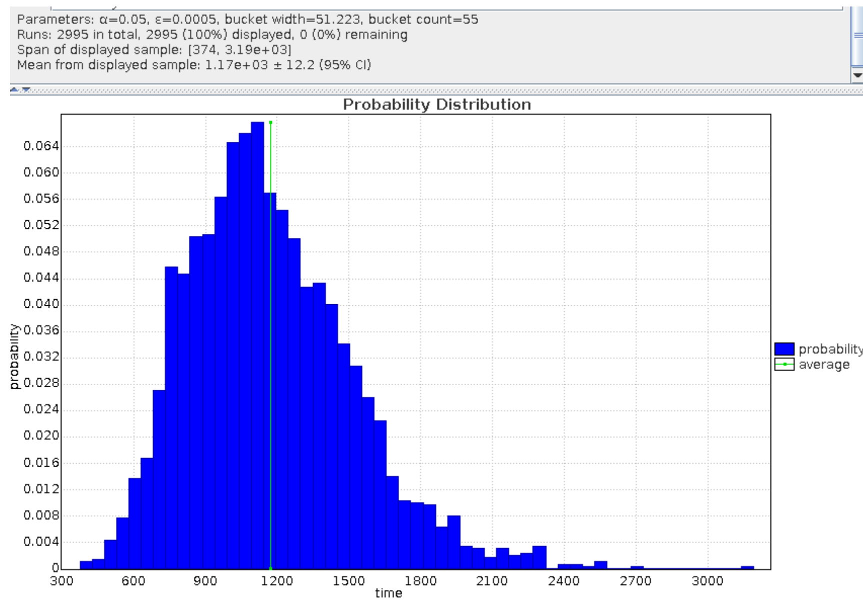
Pr [t <= 5000] (<>HeatingProcess.heat)

užklausa buvo vykdyta su atvejais, kuomet temperatūros jutiklio gedimas yra labiau tikėtinas nei veikimas ir atvirkščiai. Skirstiniai yra panašūs į ankstesnius (23-26 pav.) kai buvo tikrinama *AirConditionerProcess.cool* lokacija kuomet patalpos temperatūra buvo 35. Taip yra todėl, nes šildytuvas veikia identiškai kaip oro kondicionierius, tačiau įsijungia, kai kambario temperatūra yra mažesnė už norimą.

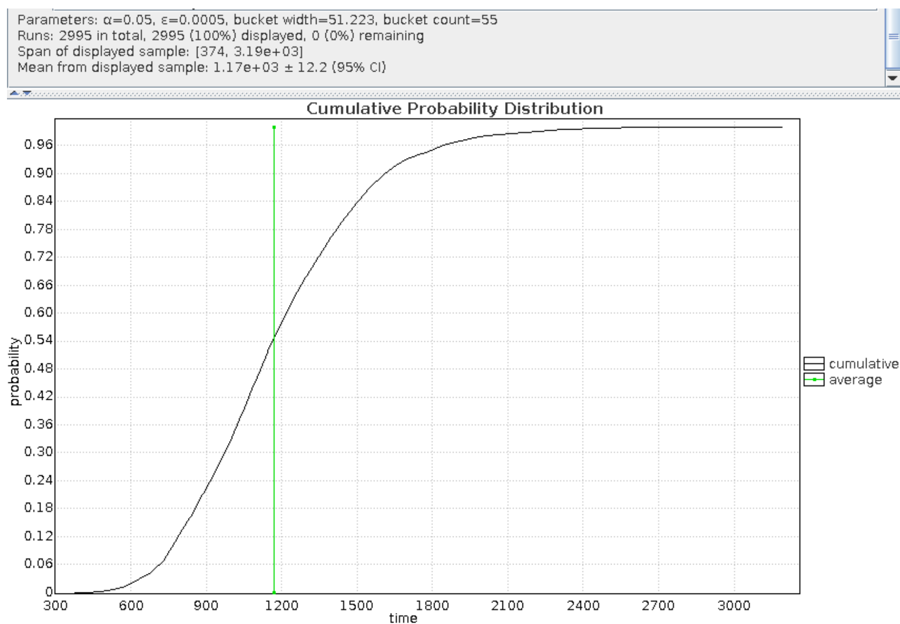
Pagrindinis visos sistemos tikslas yra pasiektas tada, kai kambario temperatūros reikšmė tampa lygi norimai temperatūros reikšmei. Tai patikrinama įvedus užklausą:

Pr[t<=10000](<>EnvironmentProcess.temperature == target_temperature)

gauti rezultatai pavaizduoti 27 ir 28 pav.



27 pav. Temperatūros pasiekimo į norimą reikšmę tikimybinis skirstinys, kai įrenginių veikimo tikimybinis svoris yra 10000, o gedimo 1



28 pav. Temperatūros pasiekimo į norimą reikšmę pasiskirstymo funkcija, kai įrenginių veikimo tikimybinis svoris yra 10000, o gedimo 1

Gauti skirstiniai parodo, kad didžiausia tikimybė, kai kambario temperatūros reikšmė taps lygi norimai temperatūros reikšmei yra tada, kai praeis maždaug 1000 laiko vienetu (vidurkis 1170). Rezultatai identiški, kai pradinė temperatūra pakeičiama į 17, kuomet šildytuvas turėtų šildyti. Minėta savybę galima verifikuoti ir kitokia formulės išraišką, pavyzdžiui:

Pr[t<=10000](<>EnvironmentProcess.temperature == target_temperature) >= 0,9

Šia formule galima užklausti, kad kambario temperatūros vertė per 10000 laiko vienetų pasieks norimą temperatūros vertę su nemažesnė nei 0,9 tikimybės verte. Pateikus šią užklausą gautas rezultatas parodė, kad sistema tenkina šią savybę. Galima mažinti laiko reikšmę ir surasti laiko ribą, ties kurią nebegalioja ši savybė. Pateikus formulę:

Pr[t<=1700](<>EnvironmentProcess.temperature == target_temperature) >= 0,9

Gaunama, kad sistema tenkina savybę, o pateikus formulę:

Pr[t<=1600](<>EnvironmentProcess.temperature == target_temperature) >= 0,9

Gaunama, kad sistema nebetenkina savybės. Rezultatų išvestis pavaizduota 29 pav.

```
Pr[t<=1700](<>EnvironmentProcess.temperature == target_temperature) >= 0.9
Verification/kernel/elapsed time used: 0.12s / 0s / 0.119s.
Resident/virtual memory usage peaks: 12,164KB / 50,760KB.
Property is satisfied.
Pr[t<=1600](<>EnvironmentProcess.temperature == target_temperature) >= 0.9
Verification/kernel/elapsed time used: 0.28s / 0s / 0.277s.
Resident/virtual memory usage peaks: 12,164KB / 50,760KB.
Property is not satisfied.
```

29 pav. Sistemos tikimybės reikšmės tikrinimas keičiant laiko apribojimą

Sistemos parametrai nuo kurių priklauso, kaip sistema veikia yra:

- Kambario temperatūra;
- Norima kambario temperatūra;
- Lauko temperatūra;
- Laikas per kurį kambarys sušyla dėl lauko vienu temperatūros vienetu;
- Temperatūros jutiklio pranešimų siuntimo periodas;
- Oro kondicionieriaus sušaldymo vienu temperatūros vienetu laikas;
- Šildytuvo sušildymo vienu temperatūros vienetu laikas;
- Įrenginio taisymo trukmė.

Dauguma iš minėtų parametrų yra pakankamai tiesiogiai susiję su sistemos veikimo greičiu, todėl buvo keičiami parametrai, kurie ne taip akivaizdžiai susiję, pavyzdžiui, sistemos pagrindinio tikslo pasiekimas buvo tikrinamas keičiant jutiklių skaičių. Tikrinta kaip sistema veikia su 1, 2, 3, 4, 5 jutikliais. Visais atvejais skirstiniai yra panašūs į 30 pav. Kai jutiklių yra 4 arba 5 tikimybės skirstinio vidurkis tapo mažesnis (1160 laiko vienetų). Keičiant taisymo trukmę nuo 10 laiko vienetų iki 50,

skirstiniai panašūs į 30 pav. Kai taisymo trukmė nustatyta buvo 40 arba 50, skirstinio vidurkis šiek tiek tapo didesnis (1180 laiko vienetu). Todėl galima teigti, kad sistema tikslą pasiekia greičiau, kai jutiklių kiekis yra didesnis ir taisymo trukmė mažesnė.

Buvo pastebėta, kad egzistuoja atvejis, kada įrenginiai visada sugenda, įskaitant atvejį, kuomet jie yra pataisomi. Todėl pateikus užklausas:

1) *EnvironmentProcess.temperature > target_temperature --> EnvironmentProcess.temperature == target_temperature;*

2) *EnvironmentProcess.temperature < target_temperature --> EnvironmentProcess.temperature == target_temperature;*

3) *AirConditionerProcess.fault or HeatingProcess.fault or TemperatureSensor1.fault or TemperatureSensor2.fault or TemperatureSensor3.fault --> not AirConditionerProcess.fault and not HeatingProcess.fault and not TemperatureSensor1.fault and not TemperatureSensor2.fault and TemperatureSensor3.fault;*

buvo grąžinti rezultatai, jog sistema netenkina šių savybių. Konkrečiau, 3) užklausa tikrina, kad jeigu bent vienas įrenginys yra sugedęs, ilgainiui visi įrenginiai bus nesugedę (pataisyti). Buvo įvestas papildomas laikrodis ir sąlyga (prielaida), kad įrenginys tam tikrą fiksuotą nedidelį laiko tarpą po sutaisymo negali sugesti. Po minėto pakeitimo 3) užklausos rezultatas buvo teigiamas.

Išvados

Šiame darbe iš analizuotų šaltinių pavyko išsiaiškinti formalaus verifikavimo, modelių patikrinimo apibrėžimus, ypatybes bei svarbą. Buvo išanalizuota, kokias sistemų savybes galima verifikuoti šiais metodais. Atskirai buvo atlikta įrankio „*Uppaal*“ apžvalga. Išanalizuota daiktų interneto sistema paimta iš [16] šaltinio, identifikuotos jos verifikuotinos savybės ir detaliau apžvelgtas tokios sistemos verifikavimas klasikiniu modelių patikrinimo būdu.

Tyrimo metu IoT sistemos modelis iš 1.2.2 skyriaus buvo pakeistas taip, kad būtų galima atlikti verifikavimą statistiniu modelių patikrinimo metodu. Visi modelio sinchronizavimosi kanalai buvo pakeisti iš paprastų į transliuojamus kanalus. Lokacijose, neturinčiose apribojimų laikui, buvo suformuluoti ir pridėti laiko invariantai bei kur prireikė, uždėtos sąlygos perėjimuose. Taip pat perėjimuose atnaujinamos laikrodžių vertės, kad neįvyktų užstingimas laiko atžvilgiu (ang. *deadlock*). Pakeisti temperatūros jutiklio, oro kondicionieriaus, šildytuvo procesai. Jiem buvo pridėti išsišakojantys perėjimai su skirtingais tikimybiniais svoriais: vienas išsišakojimas veda į įrenginio veikimą, o kitas į jo sugedimą. Įrenginių gedimams pranešti buvo įvesti skirtingi kanalai, tam kad taisytojo procesas atskirtų kuriam įrenginiui reikalingas taisymas.

Atlikus minėtus modelio pakeitimus pradėjo veikti simuliacija ir buvo priimamos SMC verifikavimo užklausos. Iš gautų rezultatų galima teigti, kad sistema veikia korektiškai, t.y. kai patalpos temperatūra didesnė už norimą – oro kondicionierius įjungtas ir šildytuvas išjungtas, o kai mažesnė už norimą – šildytuvas įjungtas, o oro kondicionierius išjungtas, sistema neužstingsta. Be to, išlieka sistemos atsparumas gedimams, veikimo stabilumas. Kambario temperatūra tampa norima maždaug po 1170 laiko vienetų, kai norimos temperatūros vertė ir esamos kambario temperatūros vertė skiriasi 9 vienetais. Papildomai buvo stebimi rezultatai keičiant jutiklių skaičių ir taisymo trukmę, kurie patvirtino, kad didėjant taisymo trukmei ir mažėjant jutilklių skaičiui, sistemai pasiekti norimą temperatūra reikia daugiau laiko.

Atlikus verifikavimą SMC metodu galima teigti, kad šis metodas parodo ne tik, ar sistema tenkina arba netenkina tam tikrą savybę, bet leidžia įvertinti sistemos veikimo efektyvumą. Be to, sistemos verifikavimas ir rezultatų analizė leidžia parinkti optimalią sistemos konfigūracija, bei aiškiai suformuluoti prielaidas, kurios reikalingos užtikrinti nuspėjamą ir patikimą sistemos darbą.

Abstract

The purpose of this thesis is to overview the statistical model checking (SMC) theory, the Uppaal tool and use them to analyze the identified properties of the chosen IoT system. Moreover, we discuss what is required to perform SMC to an Internet of Things (IoT) system model. The room temperature control system was chosen for analysis.

In the practical work part, the IoT system model, which was verified using classical model checking, was changed so that the requirements for SMC verification would be satisfied. The obtained verification results have showed that the system effectively and predictably behaves reaching its required goals. Effectiveness might be affected by the device fault probability weight, temperature sensor count, repair time length. Overall, the analysis and verification showed that the analyzed system is stable, deadlock-free, functionally correct, reaches the target temperature, fault tolerant.

Literatūra

[1] Jun Pang, Formal Vertification of Distributed Systems, Ponsen & Looijen B.V, 2004.

[2] E. A. Lee, Cyber Physical Systems: Design Challenges, *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, 2008, pp. 363-369, doi: 10.1109/ISORC.2008.25.

[3] Incki, Koray & Ari, Ismail & Sözer, Hasan, Runtime Verification of IoT Systems using Complex-Event Processing, *2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC)*, 2017, pp. 625-630 doi: 10.1109/ICNSC.2017.8000163.

[4] Katharina Hofer-Schmitz, Branka Stojanović, Towards formal verification of IoT protocols: A Review, *Computer Networks*, 174, 2020, pp. 107-233, doi: 10.1016/j.comnet.2020.107233.

[5] K. Keerthi, I. Roy, A. Hazra, C. Rebeiro, Formal Verification for Security in IoT Devices, in: Security and Fault Tolerance in Internet of Things, *Internet of Things*, Springer, 2019, pp. 179–200, doi: 10.1007/978-3-030-02807-7_9.

[6] D. Basin, C. Cremers, C. Meadows, Model Checking Security Protocols, in: Hand- book of Model Checking, Springer, 2018, pp. 727–762, doi: 10.1007/978-3-319-10575-8_22.

[7] Christel Baier, Joost-Pieter Katoen, Principles of Model Checking, The MIT Press, 2008.

[8] Uppaal, integrated tool environment for modeling (https://uppaal.org/).

[9] Bosnacki, Dragan, Enhancing state space reduction techniques for model checking. *Journal of Economic Psychology - J ECON PSYCH,* 2001.

[10] Legay, Axel, Lukina, Anna, Traonouez, Louis Marie, Yang, Junxing, Smolka, Scott A., Grosu, Radu, Statistical Model Checking, Springer, 2019, doi: 10.1007/978-3-319-91908-9_23.

[11] F. Kong, M. Xu, J. Weimer, O. Sokolsky and I. Lee, Cyber-Physical System Checkpointing and Recovery, *2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS)*, 2018, pp. 22-31, doi: 10.1109/ICCPS.2018.00011.

[12] W. Hoeffding. Probability inequalities. *Journal of the American Statistical Association*, Vol. 58, No. 301, 1963, pp. 13-30.

[13] Herault, Thomas & Lassaigne, Richard & Magniette, Frédéric & Peyronnet, Sylvain. (2004). Approximate Probabilistic Model Checking. LNCS. 2937. 73-84. 10.1007/978-3-540-24622-0_8.

[14] H. L. S. Younes, Verification and Planning for Stochastic Processes with Asynchronous Events, *PhD thesis*, Carnegie Mellon, 2005.

[15] A. Wald, Sequential tests of statistical hypotheses, *Annals of Mathematical Statistics*, 16(2), pp. 117–186, 1945.

[16] Chen G, Jiang T, Wang M, Tang X, Ji W, Design and model checking of timed automata oriented architecture for Internet of thing, *International Journal of Distributed Sensor Networks*, 2020, doi:10.1177/1550147720911008.

[17] David, A., Larsen, K.G., Legay, A. et al, UPPAAL SMC tutorial, *Int J Softw Tools Technol Transfer 17*, pp. 397–415, 2015, doi: 10.1007/s10009-014-0361-y.

[18] Larsson, Jonatan, Automatic Test Generation and Mutation Analysis using UPPAAL SMC, *Conference Proceedings*, 2017.

[19] Ehsan Khamespanah, Ramtin Khosravi, Marjan Sirjani, An efficient TCTL model checking algorithm and a reduction technique for verification of timed actor models, *Science of Computer Programming*, Volume 153, 2018, Pages 1-29, ISSN 0167-6423, doi: 10.1016/j.scico.2017.11.004.

[20] David, Alexandre & Du, Dehui & Larsen, Kim & Legay, Axel & Mikučionis, Marius & Poulsen, Danny & Sedwards, Sean, Statistical Model Checking for Stochastic Hybrid Systems. *Electronic Proceedings in Theoretical Computer Science*. 92. 10.4204/EPTCS.92.9.

[21] Clarke, Edmund M. and Zuliani, Paolo, Statistical Model Checking for Cyber-Physical Systems, *Proceedings of the 9th International Conference on Automated Technology for Verification and Analysis*, Springer-Verlag, 2011.

[22] Meng, Weizhi, Xie, Jian, Tan, Wenan, Fang, Bingwu, Huang, Zhiqiu, Towards a Statistical Model Checking Method for Safety-Critical Cyber-Physical System Verification, *Security and Communication Networks,* 2021, doi: 10.1155/2021/5536722.

[23] F. Cicirelli and L. Nigro, Model Checking Actor-based Cyber-Physical Systems, *in 2020 IEEE/ACM 24th International Symposium on Distributed Simulation and Real Time Applications (DS-RT)*, Prague, Czech Republic, 2020 pp. 1-8, doi: 10.1109/DS-RT50469.2020.9213705.

[24] Kalajdzic, K., Jegourel, C., Lukina, A., Bartocci, E., Legay, A., Smolka, S. A., Grosu, R.", Feedback Control for Statistical Model Checking of Cyber-Physical Systems, *Leveraging Applications of Formal Methods, Verification and Validation: Foundational TechniquesI,* Springer International Publishing, 2016, pp. 46-61, doi: 10.1007/978-3-319-47166-2_4.

[25] C. -C. Chan, C. -Z. Yang and C. -F. Fan, Security Verification for Cyber-Physical Systems Using Model Checking, in *IEEE Access*, vol. 9, pp. 75169-75186, 2021, doi: 10.1109/ACCESS.2021.3081587.