TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Arvutiteaduse instituut

ITI70LT

Age Kruusamägi 121870IAPM

HAJUSSÜSTEEMIDE MUDELIPÕHINE TESTIMINE TALLINNA TÄNAVAVALGUSTUSE SÜSTEEMI NÄITEL

magistritöö

Juhendaja: Jüri Vain
Doktorikraad
Professor

Tallinn 2016

Autorideklaratsioon

Kinnitan, et olen koostanud antud lõputöö iseseisvalt ning seda ei ole kellegi teise poolt varem kaitsmisele esitatud. Kõik töö koostamisel kasutatud teiste autorite tööd, olulised seisukohad, kirjandusallikatest ja mujalt pärinevad andmed on töös viidatud.

Autor: Age Kruusamägi

09.05.2016

Annotatsioon

Käesoleva magistritöö „Hajussüsteemide mudelipõhine testimine Tallinna tänavavalgustuse süsteemi näitel“ eesmärgiks on hajussüsteemide mudelipõhiste testimisvahendite ja metoodika rakendusvõimaluste uurimine Tallinna tänavavalgustuse süsteemi näitel.

Töös on tutvustatud mudelipõhist testimist ning mudelipõhise testimise keerukusi hajusate süsteemide korral. Lisaks on kirjeldatud töös kasutatud modelleerimis-, verifitseerimis- ja testimisvahendeid, milleks on UPPAAL, UPPAAL TRON ning DTRON. Töö tulemusena on valminud Tallinna tänavavalgustuse süsteemi kirjeldavad mudelid ning keskkonnas DTRON mudelipõhiseks testimiseks vajalikud testiadapterid.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 52 leheküljel, 4 peatükki ja 7 joonist.

Abstract

Model-based testing of distributed systems: Tallinn streetlight system case-study

The aim of present thesis is to study the possible implementation of model-based testing facilities and methodology based on the example of Tallinn Streetlight system.

The paper introduces the concept of model-based testing and outlines the complexities of the model-based testing in distributed systems. The testing tools UPPAAL, UPPAAL TRON and DTRON applied in the case-study are introduced. As the result of thesis, the descriptive models of Tallinn Streetlight system and the adapters required for model-based testing have been compiled in the UPPAAL modelling and model testing environment.

The thesis is in Estonian and contains 52 pages of text, 4 chapters and 7 figures.

Lühendite ja mõistete sõnastik

Determinism (*Determinism*)	Süsteemi omadus, mille korral süsteemi väljund ja järgnev olek on üheselt määratud käesoleva oleku ja sisendiga.[1]
Formaalne verifitseerimine (*Formal verification*)	Tarkvara omaduste tõestamine formaalloogika ja matemaatiliste meetoditega. Formaalse verifitseerimise käigus luuakse süsteemi formaliseeritud esitus, mille alusel on võimalik teha loogilisi järeldusi süsteemi omaduste kohta kasutades selleks algoritmilisi või deduktiivseid järeldusmehhanisme. [24]
Funktsionaalne- ehk vastuvõtutestimine (*Conformance testing*)	Testimise liik, mille eesmärgiks on kontrollida süsteemi vastavust seatud nõuetele. [20]
Hajussüsteem (*Distributed system*)	Hajussüsteem on autonoomsete arvutite kogum, mis on omavahel ühendatud arvutivõrku ja mis on varustatud integreeritud keskkonna loomiseks vajaliku tarkvaraga. [21]
Host (*Host*)	Võrku ühendatud arvuti, mis toimib võrgu suhtes informatsiooni töötleva üksuse ja info allikana. [23]
Kohalik host (*Localhost*)	Võrku ühendatud arvuti, millest edastatav info liigub vaid ühe hosti piires. [23]
Mitte-determinism (*Non-Determinism*)	Süsteemi omadus, mille korral väljund ja järgnev olek ei ole üheselt määratud käesoleva oleku ja sisendiga. [1]
Port (*Port*)	Rakendusprogrammides ja arvutites kommunikatsiooni ühendust identifitseeriv number, mille abil adresseeritakse võrgus edastatavaid andmevoogusid ja pakette, et seostada sissetulevaid andmeid vajaliku teenusega. [23]
SUT (*System under test*)	Testitav süsteem.
Valideerimine (*Validation*)	Vastavuse kontroll nõuetega.
XML (*Extensible Markup Language*)	Standardiseeritud infoesituse märgistuskeel, kus info on esitatud struktureeritud kujul.

Sisukord

Sissejuhatus 9
1 Hajussüsteemide mudelipõhine testimine 11
1.1 Mudelipõhine testimine 11
1.2 Hajussüsteemide mudelipõhise testimise keerukused 12
2 Teoreetilised alused 15
2.1 UPPAAL 15
2.1.1 UPPAAL-i modelleerimiskeel 15
2.1.2 Simulaator 17
2.1.3 Mudeli kontrollija 18
2.2 UPPAAL TRON 19
2.2.1 Testide genereerimise algoritm 20
2.3 DTRON 21
3 Tallinna Tänavavalgustuse süsteem 23
3.1 Süsteemi kirjeldus 23
3.2 Modelleerimise kitsendused 25
3.3 Mudelid 26
3.3.1 Sõnumite vastuvõtja (SUT-i) mudel 27
3.3.2 Kilbi mudel 29
3.3.3 Kilbi mall 31
3.3.4 Mudelites juhusliku arvu genereerimine 32
3.3.5 Mudelite deklaratsioonid 33
3.3.6 Mudelitele seatud nõuded 35
3.4 DTRON 37
3.4.1 Adapter 37
3.4.2 Sõnumite vastuvõtja simulaator 39
3.4.3 Testide käivitamine 39
3.4.4 Testimise tulemus 41
4 Modelleerimislahenduste võrdlev analüüs 43

4.1 Parametriseeritud mallid ja sünkroniseerimiskanalid 43
4.1.1 Kanalimassiivide kasutamise eelised ja puudused 43
4.2 Individuaalsed kanalid protsesside vahel 44
4.2.1 Modelleerimise kitsendused 44
4.2.2 Protsessidevaheliste individuaalsete kanalite eelised ja puudused 45
Analüüsi järeldused 47
Kokkuvõte 49
Kasutatud materjalid 51
Lisa 1 – Nõuete kontrolli tulemused 53
Lisa 2 – Nõuete kontrolli tulemused 54
Lisa 3 – Adapter 55
Lisa 4 – Testi tulemus 57
Lisa 5 – Testi tulemus ebaõnnestunud testiga 61
Lisa 6 – Parametriseeritud sünkroniseerimiskanalitega mudelid 64
Lisa 7 – Individuaalseid protsesside vahelisi kanaleid kasutavad mudelid 65

Jooniste loetelu

Joonis 1 Juhitavuse probleem [5] 13
Joonis 2 Jälgitavuse probleem [5] 13
Joonis 3 UPPAAL TRON ja DTRON [6] 22
Joonis 4 Tallinna tänavavalgustuse süsteem 23
Joonis 5 Sõnumite vastuvõtja (SUT-i) mudel 27
Joonis 6 Kilbi mudel 29
Joonis 7 Juhusliku arvu genereerimine 33

Sissejuhatus

Tarkvara testimine on oluline etapp tarkvara arendustsüklis. Testimise käigus kontrollitakse tarkvara vastavust seatud nõuetele ning testi tulemuste põhjal antakse arendajale tagasisidet leitud vigade iseloomust. Testide läbimisel ei saa väita, et tarkvaras ei ole vigasid, kuid testimine tõstab tarkvara usaldusväärsust. Seega on testimine tarkvara osaline korrektsuse näitamise vahend. Täieliku korrektsuse tõestamine nõuab formaalsete verifitseerimismeetodite rakendamist.

Kaasaegsete tarkvarasüsteemide integratsioonitaseme pidev suurenemine nõuab üha rohkem testimise efektiivsusele tähelepanu pööramist. Klassikaline automatiseeritud testimine ei taga alati piisavat kvaliteeti, sest selle käigus võivad paljud olukorrad kontrollimata jääda. Mudelipõhine testimine on üks automaattestimise liikidest, mille käigus testija koostab testitava süsteemi nõudeid kirjeldavad mudelid ning testlood genereeritakse koostatud mudelitest automaatselt. See tagab, et läbitud testlugude arv on suurem kui klassikalise automatiseeritud testimise korral. Lisaks mudelitele tuleb kirjeldada adapterid, läbi mille suhtlevad koostatud mudelid testitava süsteemiga.

Hajussüsteemide integratsioonitestide korral on mitmeid samaaegselt testitavaid komponente ning lisaks tuleb arvestada komponentide vahel toimuva kommunikatsiooniga. Testimise puhul on oluline, et koostatud testid ei oleks vigased ning simuleeriks võimalikult täpselt testitava süsteemi nõuetes kirjeldatud käitumist. Seega on hajussüsteemide testimisel eriti oluline jälgida sõnumite edastust, nende õiget järjekorda ja ajastamist.

Antud magistritöö eesmärgiks on hajussüsteemide mudelipõhise testimisvahendite ja metoodika rakendusvõimaluste uurimine Tallinna tänavavalgustuse süsteemi näitel. Magistritöö praktilise osa tulemusena töötatakse välja süsteemi kirjeldavad mudelid ning mudelipõhiseks testimiseks vajalikud adapterid. Antud süsteemi iseloomustab väga suur hulk üle linna asetsevaid tänavavalgusteid lokaalselt juhtivaid kontrollereid, mis on paigutatud nn kilpidesse, mis peavad keskse serverrakendusega suhtlema ning sellele lokaalse kontrolleri kohta käivat infot edastama. Lisaks võrreldakse töös erinevaid

hajussüsteemide modelleerimislahendusi modelleerimis- ja mudelkontrolli keskkonnas UPPAAL.

Töö on jaotatud neljaks osaks, millest esimeses osas antakse ülevaade mudelipõhisest testimisest ning mudelipõhise testimise keerukusest hajussüsteemide korral. Teises peatükis kirjeldatakse kasutatavaid töövahendeid, milleks on UPPAAL, *online* testimiskeskkond UPPAAL TRON ning DTRON, millest viimane on UPPAAL TRON-i edasiarendus hajussüsteemide mudelipõhiseks testimiseks. Töö kolmandas peatükis kasutatakse magistritöö kahes esimeses peatükis kirjeldatud modelleerimistehnikat ning testimisvahendeid Tallinna tänavavalgustuse süsteemi mudelipõhiseks testimiseks. Neljandas peatükis võrreldakse UPPAAL-i modelleerimiskeele ning UPPAAL TRON-i piirangutest tulenevaid alternatiivseid modelleerimislahendusi Tallinna tänavavalgustussüsteemi hajustestimiseks. Analüüsil võrreldakse mudeleid lähtuvalt nende esitustäpsuse, analüüsitavuse ja testide skaleeruvuse aspektidest.

1 Hajussüsteemide mudelipõhine testimine

1.1 Mudelipõhine testimine

Tarkvara testimine jaguneb manuaalseks ning automaatseks testimiseks. Manuaalse testimise korral koostab testija nõuete põhjal testlood ning seejärel täidab need käsitsi. Manuaalne testimine on küllaltki aeganõudev ning selleks, et kiirendada testimist ja suurendada testide hulka, tasub manuaalset testimist kombineerida automatiseeritud testimisega. Automatiseeritud testimise korral koostab testija testlood, mille põhjal programmeeritakse automaattestid, mida käivitatakse korduvalt. Automaattestimise üheks puuduseks on asjaolu, et automaattestide kirjutamine ning uuendamine, kui tarkvara või sellele seatud nõuded muutuvad, on samuti aeganõudev ning seega ei pruugi alati märkimisväärselt tarkvara kvaliteeti tõsta. Automaattestimise üks alaliikidest on mudelipõhine testimine, mis peaks vähendama testide ajakohastamisele kuluvat aega tarkvara või sellele seatud nõuete muutumisel. Mudelipõhise testimise korral kirjeldab testija süsteemi iseloomustavad mudelid. Testilood ning testiskriptid genereeritakse koostatud mudelitest automaatselt. [1]

Mudelipõhise testimise eelis manuaalse või klassikalise automatiseeritud testimise ees seisneb selles, et mudelipõhine testimine võimaldab teste otseselt siduda süsteemile seatud nõuetega. See muudab testide loetavuse, hooldamise ja nendest arusaamise lihtsamaks. Lisaks aitab mudelipõhine testimine tagada süsteemi käitumise hea katvuse testidega ning vähendada testimisele kuluvat aega. [4] Mudelipõhise testide genereerimise efektiivsus tuleneb sellest, et testid genereeritakse nõuete formaliseerimisel saadud mudelitest automaatselt. Lisaks suurendab see vigade avastamise tõenäosust, kuna süsteemi nõutava käitumise kohta loodud mudelitest saab genereerida teste süstemaatilisemalt ja suuremas mahus kui manuaalse testimise korral. Seega läbitakse testimise käigus laiem süsteemi funktsionaalsus, mis peaks suurendama tarkvara usaldusväärsust. [8]

Mudelipõhise lähenemise eeliseks on ka testide modifitseerimise lihtsus, sest nõuete muutumise korral tuleb muuta üksnes mudelit, mille põhjal saab testid uuesti

genereerida. [7] Selle eelduseks on loomulikult mudelite loetavus ning ühene arusaadavus. Seega ei tohiks süsteemi piisaval tundmisel mudelite muutmine olla keeruline ka teistele süsteemi testijatele. Selline lähenemine peaks lühendama testide hooldamisele kuluvat aega, sest klassikalise automaatse testimise üheks suurimaks kitsaskohaks on testide hooldamine ning sellele kuluv aeg.

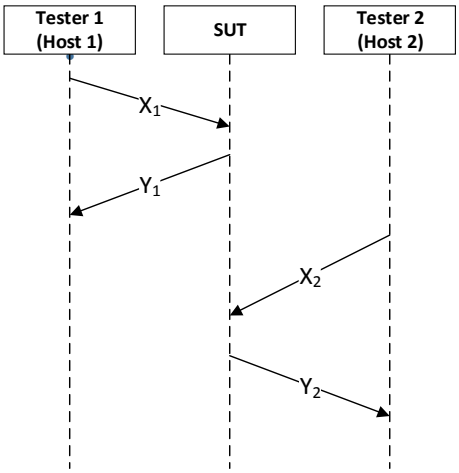
Teiseks suurimaks mudelipõhise testimise kitsaskohaks on testiadapterite loomine, mida on suhteliselt keeruline automatiseerida. Adapter on vahelüli, läbi mille suhestuvad abstraktne mudel ning konkreetne testitav süsteem. Kui tarkvarale seatud nõuded muutuvad, siis lihtsamal juhul saab muuta vaid mudelit, millest saab kergesti genereerida uued testid. Testiliidese muutumisel tuleb lisaks mudelile muuta ka adapterit. [1]

Mudelipõhine testimine toimub üldjuhul „musta-kasti“ põhimõttel, kus testitava süsteemi käitumine on jälgitav ainult väljundite kaudu ja mõjutatav testsisendite kaudu. See tähendab, et testide koostamisel vaadeldakse ja analüüsitakse ainult sisendite-väljundite käitumist. [2] Seega ei pea mudelite koostamisel teadma süsteemi realiseerimise nüansse ja sisemist ülesehitust. Piisab liideste ja ilmutatud funktsionaalsuse tundmisest ning mudeleid saavad koostada testijad, kes ei ole tarkvara koodi autorid.

1.2 Hajussüsteemide mudelipõhise testimise keerukused

Hajussüsteemide mudelipõhisel testimisel on kaks peamist keerukust, mis on kirjeldatud alljärgnevalt.

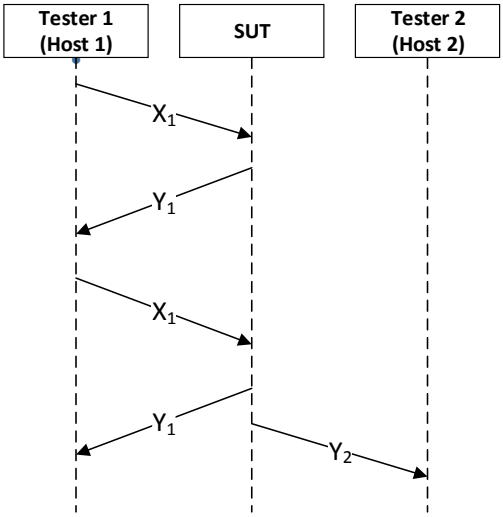
- **Juhitavuse probleem** ehk kui süsteemis on mitu hosti ning tegevused erinevatel hostidel peavad rangelt järgnema üksteisele ja mitte toimuma samaaegselt, siis kuidas saada teada millal tegevus eelneval hostil on lõppenud. Juhitavuse probleemi illustreerib joonis 1.



Joonis 1 Juhitavuse probleem [5]

Eeldame, et süsteemis on kaks hosti (Host 1 ja Host 2), milledel asuvad testerid. Tegevused peavad toimuma järjekorras, kus hosti Host 1 sisend X_1 viib väljundini Y_1 ning sellele peab järgnema sisend X_2 hostis Host 2, mis viib väljundini Y_2. Kui süsteemis on kaks hosti ning tegevus teises hostis peab toimuma alles siis kui esimese hosti tegevus on lõppenud, seisneb testimise keerukus selles, et teises hostis (Host 2) asuv tester ei tea, millal esimeses hostis (Host 1) on tegevus lõppenud.

- **Jälgitavuse probleem** ehk kuidas tagada, et tegevuste järjekord on õige, kui SUT tagastab küll õiged väljundid, kuid need läbitakse vales järjekorras. Jälgitavuse probleemi illustreerib joonis 2.



Joonis 2 Jälgitavuse probleem [5]

Eeldame, et süsteemis on kaks hosti (Host 1 ja Host 2), milledel asuvad testerid. Testi tegevused peavad toimuma järjekorras, kus esmalt esimese hosti (Host 1) sisend X_1 peab viima väljundini Y_1, millele järgneb samas hostis (Host 1) uuesti sisend X_1, mis viib väljundini Y_1. Esimeses hostis (Host 1) asuv tester (Tester 1) jälgib sisendite ja väljundite kombinatsiooni $X_1Y_1X_1Y_1$ ja teises hostis (Host 2) asuv tester (Tester 2) jälgib ainult väljundit Y_2. Testimise keerukus seisneb sisend/väljund tegevuste õige järjekorra tagamises. Sisendile X_1 peab esmalt järgnema ainult väljund Y_1 ning olukord, kus sisendile X_1 järgnevad esmalt kaks väljundit Y_1 ja Y_2 võib viia vigadeni. [12]

Hajusate süsteemide testimine on põhiliselt suunatud testide sisendite ja väljundite järjestuste leidmisele nii, et testi juhitavus ja jälgitavus oleksid tagatud.

2 Teoreetilised alused

2.1 UPPAAL

UPPAAL on Uppsala ja Aalborgi ülikoolide koostöös välja töötatud modelleerimis- ja mudelkontrolli keskkond, mille põhilised komponendid on graafiline kasutajaliides reaalajasüsteemide modelleerimiseks, simulaator ja mudelkontrollija. [3] Süsteemi nimetatakse reaalajasüsteemiks, kui süsteemis tehtavate operatsioonide õigsus sõltub lisaks tehtavate operatsioonide loogilisele õigsusele ka nende täitmise ajast. [10]

UPPAAL põhineb ajaga automaatide teoorial. Ajaga automaatide süntaksi ja semantikat kirjeldavad järgmised definitsioonid:

Definitsioon 1: *Ajaga automaat on korteez* (L, l_0, C, A, E, I)*, kus L on seisundite hulk,* $l_0 \in L$ *on algseisund, C on mudeli kellade hulk, A on mudeli tegevuste, kaastegevuste ja sisemiste τ-tegevuste hulk,* $E \subseteq L \times A \times B(C) \times 2^C \times L$ *on süsteemi seisunditevaheliste siirete hulk, koos tegevuste, siirete valvurite ja algväärtustusega kellade hulgaga. Kujutus* $I : L \rightarrow B(C)$ *seab seisundite hulgale vastavusse invariantide hulga.* [9]

Definitsioon 2: *Ajaga automaadi* (L, l_0, C, A, E, I) *semantika on üleminekusüsteem* $\langle S, s_0, \rightarrow \rangle$*, kus* $S \subseteq L \times \mathbb{R}^C$ *on süsteemi olekute hulk,* $s_0 = (l_0, u_0)$ *on süsteemi algolek ja* $\rightarrow \subseteq S \times (\mathbb{R}_{\geq 0} \cup A) \times S$ *on üleminekuseosed nii, et:*

- $(l, u) \xrightarrow{d} (l, u + d)$ *kui* $\forall d' : 0 \leq d' \leq d \Rightarrow u + d' \in I(l)$ *ja*
- $(l, u) \xrightarrow{a} (l', u')$ *kui eksisteerib* $e = (l, a, g, r, l') \in E$ *s.t.* $u \in g$, $u' = [r \mapsto 0]u$, *ja* $u' \in I(l')$,

kus $d \in \mathbb{R}_{\geq 0}$, $u + d$ *tähistab kellade väärtustust, kus väärtusele u on liitunud hilistumine d ja* $[r \mapsto 0]u$ *tähistab kellade väärtustust, kus kellade hulga C alamhulga r väärtused on nullitud.* [9]

2.1.1 UPPAAL-i modelleerimiskeel

UPPAAL-i mudeleid kirjeldatakse kui ajaga automaatide võrgustikke, kus iga automaat koosneb seisunditest (*Locations*) ja seisunditevahelistest siiretest (*Edges*).

Seisundid esitatakse graafiliselt ringidena ning igale seisundile antakse selle identifitseerimiseks nimi. Seisunditele saab määrata invariandid ehk tingimused, mis peavad olema täidetud selleks, et süsteem saaks olla antud seisundis.

Seisundi tüüp võib olla:

- **Algseisund** (*initial location*) määrab süsteemi alustamise seisundi. Algseisund on mudelil märgitud topeltringiga ning igal automaadil saab olla vaid üks algseisund.
- **Kiireloomuline seisund** (*urgent location*) on semantiliselt samaväärne seisundiga, kus süsteemi kell x alglähtestatakse (x = 0) seisundisse jõudmisel ning seisundi invariant on $x \leq 0$. Kui süsteem on kiireloomuliseks märgitud seisundis, siis süsteemikell sellel ajal edasi ei liigu.
- **Hetkeline seisund** (*committed location*) on sarnane kiireloomulisele seisundile ehk kui süsteem on hetkelises seisundis, siis süsteemikell edasi ei liigu. Erinevus kiireloomulisest seisundist seisneb selles, et kui süsteem on hetkelises seisundis, siis järgmisena saab süsteem läbida vaid siiret, mis algab sellest seisundist. Teiste paralleelsete automaatide täitmine on seni blokeeritud. [9]

Seisundid on ühendatud siiretega ning siirete atribuudid on alljärgnevad:

- **Mitte-deterministlik omistamine** (*select*) esitatakse kujul *muutuja nimi : tüüp*, mis tähistab tüübi mitte-deterministlikult valitud väärtuse omistamist nimega esitatud muutujale. Seejuures muutujad on deklareeritud siirdel, kus nende väärtus omistatakse.
- **Valvur** (*guard*) väljendab tingimust, mis peab olema täidetud selleks, et süsteem saaks siiret läbida.
- **Sünkroniseerimise** (*synchronisation*) väljal kirjeldatakse kanalid, läbi mille paralleelsete automaatide siirete täitmist sünkroniseeritakse. Kanalite sünkroniseerimise suuna määrab kanali nime sufiks „?“ või „!“.
- **Omistamise** (*update*) väljal kirjeldatakse muutujate omistamist, mis täidetakse siirde läbimisel. Omistamistingimustega saab spetsifitseerida alternatiivseid omistamisi vastavalt sellele, kas tingimus kehtib või mitte. [9]

2.1.2 Simulaator

Simulaatorit kasutatakse koostatud mudelite täitmise visuaalseks jälgimiseks. Peamiselt on simulaator abiks mudeli koostajale, sest selle abil saab mudeleid käivitada ning jälgida nende käitumist.

UPPAAL-i simulaatori vaade koosneb neljast paneelist:

- **Simulatsiooni juhtimispaneelil** (*Simulation control*) käivitatakse mudeleid ning seda osa kasutatakse simulatsiooni andmete kuvamiseks. Juhtimispaneel võimaldab valida mudeli järgmisena täidetavaid siirdeid ja valida täitmisrežiime. Üheks režiimiks on samm-sammuline mudelite läbimine ja mudelite käitumise jälgimine sellel ajal. Simuleerimisel tõstetakse erineva värviga esile jooksval täitmissammul aktiivsed seisundid. Simulatsiooni paneelil kuvatakse kõiki siirdeid, mis on mudelites jooksval täitmissammul lubatud. Kuna ühest seisundist väljuvaid siirdeid võib olla rohkem kui üks, siis kuvatakse selles osas kõiki siirdeid, mida oleks võimalik järgmisena läbida.
- **Muutujate** (*Variables*) vaates kuvatakse muutujate ja kellade väärtusvahemikke antud simulatsiooni sammul. Kuvatavad muutujad valitakse *View* rippmenüüst ning nende väärtused kuvatakse seepeale simulaatori muutujate paneelil. Juhtpaneel võimaldab valida ka seisundeid tagasiulatuvalt simulatsioonilogi eelmistest sammudest. Kui logist valitakse siire, siis muutujate osas kuvatakse muutujate väärtused siirdele eelnevas seisundis, mille korral on valitud siirde täitmine lubatud.
- **Protsesside** (*Processes*) vaates kuvatakse graafiliselt mudeli mallide eksemplare, mida nimetatakse protsessideks. Protsesside vaade on visuaalselt hästi jälgitav, sest aktiivsed seisundid ehk need seisundid, milles süsteem hetkel on ning siirded, mida järgmisena läbitakse, on märgitud punaselt. Kui simulatsiooni juhtimispaneelil valitakse aktiivsed seisundid või seisunditest väljuvad siirded, siis protsesside vaates tähistatakse neid punasena. Protsesside vaadet saab muuta vastavalt sellele, milliseid protsesse soovitakse antud hetkel jälgida. Seega saab selles vaates näiteks osa protsesse ajutiselt peita.
- **Sõnumite jadadiagrammi** (*Message sequence chart*) paneelil on iga protsessi kohta vertikaalne ajatelg ning läbitud mudeli seisundeid kuvatakse nendel horisontaalsete ridadena. Lisaks kuvatakse seal ka mudelite vahel edastatud

sünkroniseerimissõnumeid. Sõnumite jadadiagramm visualiseerib protsesside interaktsioone simulatsiooni ajateljel ja selle abil saab testija valida erinevaid horisontaalseid ridasid, mille peale uuendatakse ka teised simulaatori vaated. Nii saab läbida ning jälgida mudelite käitumist ja seisundeid erineva detailsusega. [17]

2.1.3 Mudeli kontrollija

Mudeli kontrollija peamine eesmärk on tõestada, et koostatud mudelid rahuldavad teatud omadusi. Testide käivitamisel leitud vigade analüüsimise juures on täiendav ajakulu, kui testid ebaõnnestusid põhjusel, et koostatud mudelid olid vigased. Lisaks on oluline tagada, et mudelid oleksid korrektsed, sest vigaste mudelite kasutamise korral võivad testimise tulemused olla mitte-usaldusväärsed.

Kirjeldatavad nõuded jagunevad kolmeks:

- **Saavutatavuse omadused** (*Reachability Properties*) on kõige lihtsam nõuete vorm ning need väljendavad seda, et mudelis leidub vähemalt üks täitmisjada, mille läbimisel on kirjeldatud nõue täidetud. Saavutatavuse nõuded tasub defineerida juba enne esmaste mudelite loomist. Nii aitavad need juba varajases mudeli loomise etapis tagada mudelite õigsuse. [15]

Saavutatavuse nõuded kirjeldatakse kujul, kus ϕ kirjeldab seisundeid, mis tuleb saavutada ja E<> temporaalne operaator, et vähemalt ühes jadas on need seisundid esindatud:

```
E<> ϕ
```

- **Ohutusomadused** (*Safety Properties*) väljendavad, et kirjeldatud omadus kehtib kõikide täitmisjada korral kogu nende jadade läbimise jooksul. Ohutusomadustega väljendatakse, et mudelis ei esine kunagi soovimatuid seisundeid.

Ohutusomadused kirjutatakse kahel erineval viisil, kus ϕ on seisundi omadust esitav valem:

```
A[] ϕ,
```

millega väljendatakse, et mudeli kõigi täitmisjadade kõigil seisundites on kirjeldatud tingimus täidetud. Teine turvalisuse nõuete kirjeldamise viis:

```
E[] ϕ,
```

millega väljendatakse, et mudelis eksisteerib vähemalt üks täitmisjada, mille läbimise jooksul on tingimus täidetud.

- **Elususomadused** (*Liveness Properties*). Kuigi ohutusomadused kirjeldavad millised tegevused ei tohi mudelis lubatud olla, siis ainult nende kirjeldamine ei ole mudeli õigesti toimimise kontrollimiseks piisav. Lisaks tuleks kirjeldada elususomadusi, mis väljendavad, et tingimus peab mudelis olema kunagi garanteeritult saavutatav. [15]

Elususe nõuded kirjeldatakse kahel erineval viisi, kus ϕ on nõude kirjeldus:

```
A<> ϕ,
```

millega väljendatakse, et mudeli igas täitmisjadas peab mingil ajahetkel kirjeldatud tingimus(ϕ) olema täidetud. Teine elususe nõuete kirjeldamise viis:

```
ψ--> ϕ,
```

väljendab seda, et seisunditele, kus kehtib ψ, järgneb vähemalt üks võimalik täitmisjada, milles esineb seisund, kus kehtib ϕ. Ehk teisisõnu, mudelis igas täitmisjadas, mis algab seisundist, kus tingimus ψ on täidetud, peab jõudma mingil ajahetkel sellisesse seisundisse, kus tingimus ϕ on täidetud.

2.2 UPPAAL TRON

UPPAAL TRON (*Testing Realtime systems ONline*) ehk edaspidi lihtsalt TRON, on testide täitmise keskkond, mis põhineb UPPAAL-il ning on mõeldud reaalajasüsteemide mudelipõhiseks musta-kasti testimiseks. [14] TRON on *online* testimisvahend, mis tähendab, et testi sisendeid genereeritakse sõltuvalt SUT-i seisundist ja testi eesmärgist

jooksvalt. TRON täidab kahte funktsiooni – nõuete mudeli simulatsioon ja SUT käitumise monitoorimine. TRON simuleerib testitava süsteemi käitumist tulenevalt nõuete mudelitest ning samal ajal jälgib testitava süsteemi käitumist ja kontrollib, kas tegeliku süsteemi väljundid vastavad mudelis kirjeldatud väljunditele. [18]

TRON-i võimalused on kirjeldatud alljärgnevalt:

- TRON on sobilik funktsionaalseks- ehk vastuvõtutestimiseks, mis kontrollib, kas testitav süsteem käitub samaselt nagu on kirjeldatud mudelites.
- Põhirõhk on aja- ning funktsionaalsete omaduste testimisel, kus ajakitsendused on väljendatavad kellade kitsenduste abil. Süsteemi sisend- ja väljundsõnumite edastus võib toimuda erinevatel ajahetkedel, kuid peab vastama seisundi invariantide ja siirete valvurite tingimustele.
- TRON on testimiseks suuteline kasutama erinevaid mudeleid. See tähendab, et mudelid võivad olla nii deterministlikud kui mitte-deterministlikud, lisaks on lubatud mudeli mallide erinevad eksemplarid ja aja/kellade seadistused. [6]

2.2.1 Testide genereerimise algoritm

Lisaks testide genereerimisele, töötab UPPAAL TRON ka testi oraaklina ehk jälgib testitava süsteemi ning mudelite sisend-väljund kombinatsioone ja kontrollib nende omavahelist vastavust. Otsustusstrateegia, mille alusel TRON genereerib testi järgnevad sammud, jaguneb kolmeks ning testi järgnev samm valitakse alljärgnevate tegevuste hulgast mitte-deterministlikult. [13]

- Seisundite hulgast Z valitakse järgmine võimalik seisund ja sellesse viiva siirdega määratud sisend saadetakse testitavale süsteemile. Kui testitav süsteem läheb uude seisundisse, uuendatakse süsteemi võimalike järgmiste seisundite hulk Z ($Z := Z$).

```
Saada sisend testitavale süsteemile:
 Kui Mudeli_Väljundite_Hulk(Z) = ∅
   Vali mitte-deterministlikult:
      i ∈ Mudeli_Väljundite_Hulk(Z)
   Saada i SUT-ile, Z := Z peale i
```

- Valida mitte-deterministlikult viivitus d ∈ Viivitused(Z). Seejärel ootab süsteem d ajaühikut või väljundi o ilmnemisel jätkab mudeli täitmist. Kui ilmneb väljund o, mis ei kuulu hulka Z (o ∉ Z), teavitatakse testi ebaõnnestumisest.

```
Viivita ja oota testitavalt süsteemilt väljundit:
 Vali mitte-deterministlikult d ∈ Viivitused(Z)
 Oota d ajaühikut või ärka väljundi o ilmnemise peale(kui d' ≤ d)
 Kui o ilmneb, siis
   Z := Z peale viivitust d
   Kui o ∉ Testitava_Süsteemi_Väljudnite_Hulk(Z) siis teavita testi
            ebaõnnestumisest
   Vastasel juhul Z := Z peale väljundit o
 Vastasel juhul Z := Z peale viivitust d
```

- Ühendus testitava süsteemiga taaskäivitatakse, see tähendab, mudelid viiakse algseisunditesse. Taaskäivitamise tõenäosus on reaalse rakenduse testimise korral viidud võimalikult madalaks, et vältida mitte-informatiivseid testisamme.

```
Alglähtesta ja taaskäivita:
 Z := {(s0, e0)}, taaskäivita SUT
```

Ootamatu süsteemi käitumine või väljundi puudumine avastatakse juhul kui järgmiste võimalike seisundite hulk Z on tühi, mis tähendab, et SUT-i käitumine ei vasta spetsifikatsioonile. [13]

```
Kui Z = ∅ teavita testi ebaõnnestumisest, vastasel juhul testi
õnnestumisest.
```

2.3 DTRON

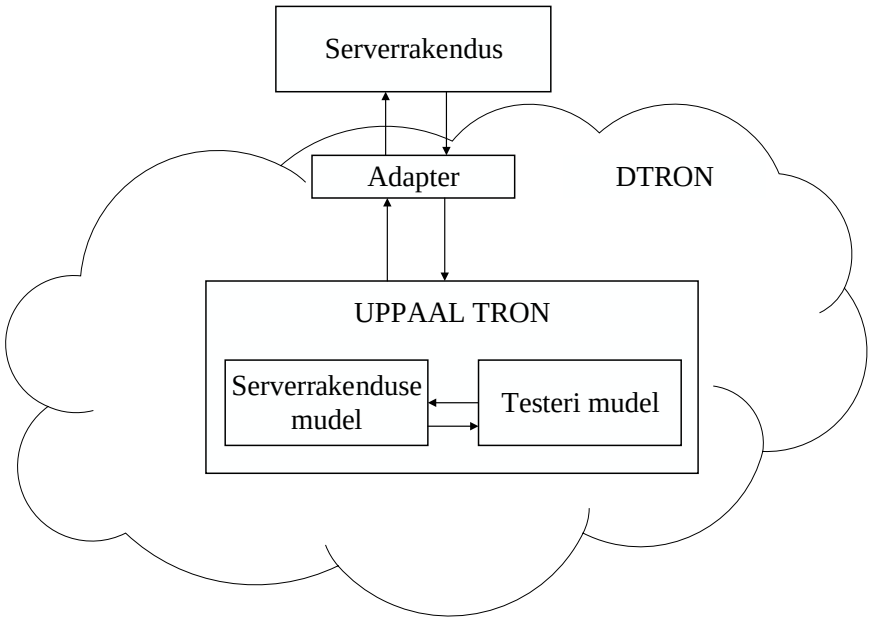
DTRON (*Distributed Testing Realtime systems ONline*) on käsurearakendus, mis põhineb UPPAAL TRON-il, kuid erinevalt TRON-ist, on mõeldud kasutamiseks hajussüsteemide mudelipõhiseks testimiseks. DTRON on raamistik, mis on ehitatud UPPAAL TRON-i ümber, et toetada paralleelset sõnumiedastust ehk edastada sõnumeid ühe adressaadi asemel mitmele. DTRON on võimeline katkestama ühelt testerilt tulevaid sõnumeid ning informeerima teisi süsteemi osasid toimunud muudatustest. Sõnumivahetus ning suhtlus testerite vahel on implementeeritud nii, et testerid saavad

lahkuda sõnumivahetusest ning liituda sõnumivahetusega ilma süsteemi infrastruktuuri uuesti konfigureerimata. [16]

DTRON-i kasutamiseks tuleb esmalt koostada UPPAAL-i modelleerimis- ja mudelkontrolli keskkonnaga mudelid, mis on viidatavad DTRON-i käivitamisel ühe sisendparameetrina. Peale DTRON-i käivitamist, otsib see esmalt kanaleid, mille kaudu koostatud mudelid suhtlevad. Kanalite nimedel peavad olema prefiksid „i_“ ja „o_“, kus prefiksit „i_“ kasutatakse testrist testitavasse süsteemi andmete edastamiseks ning prefiksit „o_“ kasutatakse testitavast süsteemist testrisse andmete edastamiseks. Mudelite suhtlus toimub kasutades Spread serverit [19], mis peab DTRON-i töötamiseks olema käivitatud. Spread server võimaldab sõnumiedastust samaaegselt mitmele saajale.

Spread server tuleb enne kasutamist konfigureerida ehk määratleda host-i aadress ning port. Lisaks on Spread serveri konfiguratsioonifailis võimalik määratleda täiendavaid tingimusi, mille hulgas on näiteks see kas logi kirjutatakse konsoolile või salvestatakse eraldi faili (*EventLogFile*) ning logimise detailsus (*DebugFlags*).

UPPAAL TRON-i ja DTRON-i omavahelised seosed on kujutatud joonisel 3. UPPAAL TRON-i ja serverrakendust liidestab adapter, mis teisendab kanalitega määratud sümbolsisendid konkreetseteks testitava süsteemi sisenditeks ning süsteemi väljundid tagasi testimudeli kanalite sümbolväljunditeks. [6]

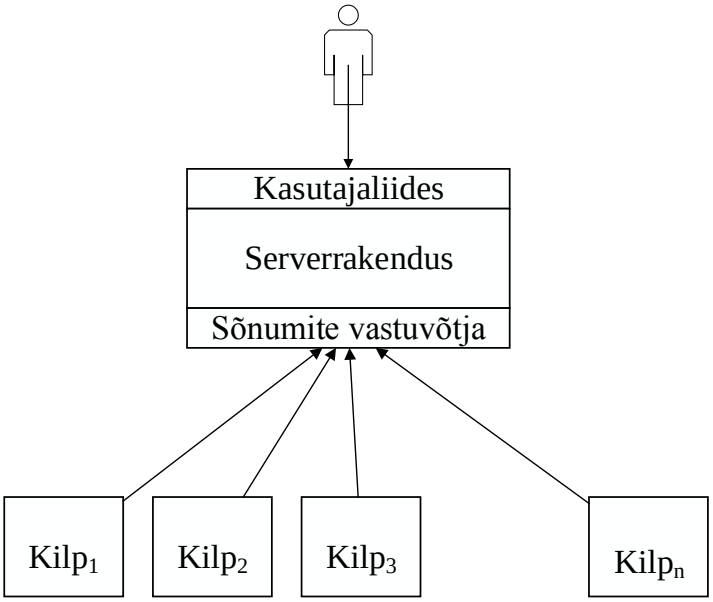


Joonis 3 UPPAAL TRON ja DTRON [6]

3 Tallinna Tänavavalgustuse süsteem

3.1 Süsteemi kirjeldus

Käesoleva magistritöö praktilises osas on käsitletud Tallinna tänavavalgustuse süsteemi sõnumite vastuvõtja ning kilpidevahelise sõnumiedastuse mudelipõhist testimist. Sõnumitega edastatakse kilpides toimunud muudatuste info serverrakendusele. Tallinna tänavavalgustuse süsteem koosneb serverrakendusest, mis hõlmab kasutajaliidest ning sõnumite vastuvõtjat. Sõnumite vastuvõtja külge ühenduvad üle Tallinna asuvad kilbid, et edastada nendes toimunud muudatuste info kesksele serverrakendusele. Serverrakenduse sisemist loogikat ning kasutajaliidest käesolevas töös ei vaadelda. Serverrakendus suhtleb kilpidega läbi traadita sideühenduse.[11] Lihtsustatult illustreerib Tallinna tänavavalgustuse süsteemi joonis 4.



Joonis 4 Tallinna tänavavalgustuse süsteem

Kilbid võtavad serverrakendusega ühendust iga 15-minutilise intervalli järel ning edastavad kilbis toimunud muudatuste info sõnumite vastuvõtjale. Kui sõnumite vastuvõtjaga ei ole antud ajahetkel võimalik ühendust luua, kordab kilp tegevust ning proovib uuesti ühendust luua seni kuni see õnnestub. Kui ühendumine ning andmete edastamine ei õnnestu 15-minutilise intervalli jooksul, edastatakse serverrakendusele

järgneval ühendumise tsüklil eelmise tsükli edastamata jäänud ning uued kogutud andmed.

Peamiseid põhjuseid, miks kilp ei saa serverrakendusega ühendust luua:

- Sõnumite vastuvõtja maksimaalne lubatud ühenduste arv on täis, sest serverrakendusega on tulenevalt tehnilistest piirangutest võimalik maksimaalselt luua 1000 ühendust. Hetkel on süsteemis umbes 600 kilpi, kuid tulenevalt kilpide arvu võimalikest muutustest, peab süsteem käituma tõrgeteta ka suurema kilpide arvu korral;
- Serverrakendus on hooldustöödel või tehniliste rikete tõttu kättesaamatu mõnele kilbile.

Suhtluse serverrakenduse ning kilbi vahel algatab kilp. Kui ühendumine õnnestub, siis kilp edastab andmed sõnumite vastuvõtjale. Kui sõnumite vastuvõtjale saabub sõnumi lõputähis, suletakse ühendus serveri ja kilbi vahel. Võib juhtuda ka, et ühendus küll luuakse ning andmeid hakatakse edastama, kuid edastamine jääb pooleli, sest kilp või serverrakendus muutuvad teineteisele kättesaamatuks. Sellisel juhul poolikult edastatud andmete paketi saatmine tühistatakse ning edastatakse terviklikult järgneval ühendumisel uuesti.

Kilbi ja serverrakenduse vahel liiguvad andmed baitide jadana. Kui sõnumite vastuvõtja saab sisendi, siis loetakse andmeid kuni saabub sõnumi lõpu tähis (*end marker*). Kui andmed on loetud, antakse need andmebaasile edasiseks töötlemiseks. Sõnum koosneb alampakettidest ehk väljadest, kus esmalt edastatakse välja identifikaator ning seejärel selle väärtus. Sõnumi lõpetab lõpu tähis.

Sõnumi struktuur:

Kilbi identifikaatori tähis	Kilbi identifikaator	Kilbi oleku tähis	Kilbi olek	Lõpu tähis

Kui serverrakendus on kilbilt andmed saanud ning hakkab neid edasi töötlema, siis serverrakendus kilbile ühtegi kinnitust andmete kohalejõudmise kohta ei saada. Küll aga saab kilp serverrakenduselt küsida konkreetse kilbi olekut ning selle põhjal on võimalik

mudelis kontrollida, kas andmed on korrektselt kohale jõudnud. Kui kilp küsib serverrakenduselt kilbi olekut, siis vastusena edastab serverrakendus andmed samasugusel baitide jada kujul nagu edastab kilp serverrakendusele andmeid.

3.2 Modelleerimise kitsendused

UPPAAL TRON on loodud UPPAAL-i ühe varasema versiooni baasil, mille funktsionaalsus ei toeta uuematele versioonidele lisatud parametriseeritud sünkroniseerimiskanaleid. Suure mudelite eksemplaride ning sünkroniseerimiskanalite arvu korral võimaldavad uuemad UPPAAL-i versioonid defineerida parametriseeritud malle, kus nurksulud nii mudeli eksemplarinimes kui sünkroniseerimiskanalites tähendavad, et tegemist on kanalite massiiviga ning nurksulgudes olev parameeter viitab massiivi elemendile. See tähendab, et kõik kilbi mudeli eksemplarid suhtlevad sõnumite vastuvõtjaga läbi erineva sünkroniseerimiskanali kuid sünkroniseerimiskanalite kohta piisab ühe adapteri programmeerimisest, sest sünkroniseerimiskanalid käituvad identselt. Samuti piisab ühe mudeli malli kirjeldamisest, sest ka kilbid käituvad identselt.

- Mudelite eksemplarid (*Gateway*[1], *Gateway*[2], *Gateway*[3],...);
- Sünkroniseerimiskanalid (*i_send*[1], *i_send*[2], *i_send*[3],...).

Tallinna Tänavavalgustuse infosüsteemis on hetkeseisuga umbes 600 kilpi. Kõik kilbid käituvad sisuliselt identselt ning kõikide kilpide ükshaaval modelleerimine ja defineerimine oleks liiga suur ning ebaotstarbekas töö. Samuti oleks sellisel juhul mudelite enda kvaliteedi tagamine väga keeruline. Kui mudeleid ning nendevahelisi sünkroniseerimiskanaleid on väga palju, muudab see adapteri programmerimise ebaotstarbekaks, sest koodi ridade arv kasvab väga kiiresti. Lisaks oleks sellisel juhul mudelite ning adapterite haldamine süsteemi nõuete muutumisel väga keeruline. Sellisel juhul ei annaks mudelipõhine testimine soovitud tulemust st ei optimeeriks testimisele kuluvat aega ega tagaks paremat testitava süsteemi kvaliteeti.

Seega mudelite konstrueerimisel on otstarbekas kasutada parametriseeritud malle:

- Tegeliku kilpide koguhulga asemel luuakse kilbi automaadi mall ning mudelite käivitamisel genereeritakse automaatselt vajalik arv kilbi eksemplare, mis

esitavad antud testijuhu puhul sisendeid serverrakendusele. Iga kilbi malli eksemplaril on unikaalne identifikaator.

    - Näiteks on kilbi malli eksemplarid: Gateway[1], Gateway[2], Gateway[3],...

- Kõik kilbi automaadi mallid kasutavad ühte sünkroniseerimiskanalit, millega käib koos globaalne muutuja (*i_send_noOfGateway*), mis antakse serveri protsessile eristamaks kilpe. Globaalne muutuja väärtustatakse konkreetse kilbi identifikaatoriga vahetult enne sünkroniseerimist serveri ja konkreetse kilbi vahel.
    - Sünkroniseerimiskanalid on i_send, o_response.
- Tallinna Tänavavalgustuse serverrakendus võimaldab kilpidelt korraga vastu võtta maksimaalselt 1000 ühendust, kuid loodud kilpide eksemplaride arv on 600. Kilpide koguarv võib muutuda ning süsteem peab tõrgeteta töötama ka juhul kui süsteemiga soetud kilpide arv ületab 1000 piiri. Seega arvestatakse mudelite modelleerimisel võimalusega, et kilpide koguarv võib muutuda suuremaks kui lubatud maksimaalne ühenduste arv. Kilpide koguarv ning maksimaalne lubatud ühenduste arv määratakse mudelites parameetritena, mille muutmisel saab testidega katta ka olukorrad, kus ühenduste arv ületab maksimaalse lubatud ühenduste piiri ning parameetrite väärtusi on võimalik testimise jooksul korduvalt muuta.
- Testi mudelis on serverrakenduse ehk SUT-i automaadi üks eksemplar. See tähendab, et kõik kilbi automaadi eksemplarid suhtlevad ühe SUT-i protsessiga. Kilbiprotsessid ning serverrakenduse protsess moodustavad läbi sünkroniseerimiskanalite omavahel suhtlevate automaatide paarid.

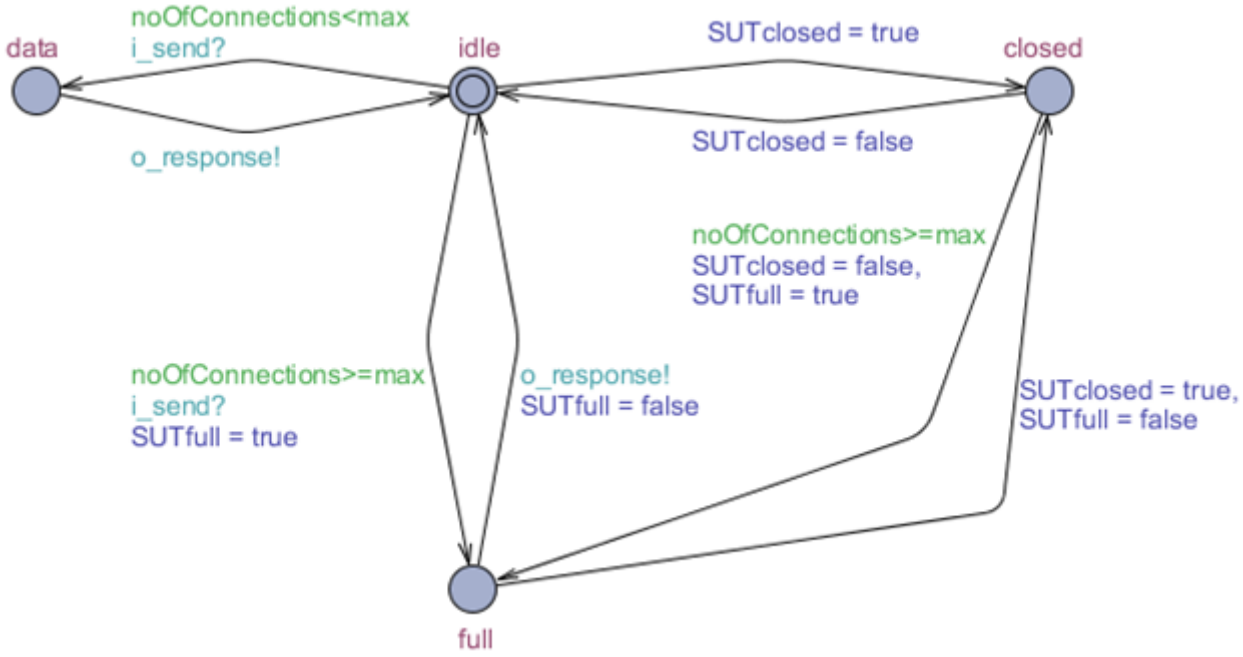
3.3 Mudelid

Tallinna Tänavavalgustuse infosüsteemi mudelipõhiseks testimiseks loodud mudelid koosnevad paralleelselt komponeeritud protsessidest, mis kirjeldavad sõnumite vastuvõtja (SUT) ning kilpide käitumist. Mudelid on koostatud UPPAAL-i modelleerimis- ja mudelkontrolli keskkonnaga, mis on kirjeldatud käesoleva magistritöö peatükis 2.1.

3.3.1 Sõnumite vastuvõtja (SUT-i) mudel

Sõnumite vastuvõtja automaat kujutab serverrakenduse käitumist kilpidelt info vastuvõtmisel läbi sünkroniseerimiskanali *i_send* ning kilbile info saatmisel läbi sünkroniseerimiskanali *o_response*. Sünkroniseerimiskanalitega koos edastatakse kilbi identifikaator. Nii teab sõnumite vastuvõtja, milliselt kilbilt infot edastatakse või millisele kilbile vastuseks info edastada. Lisaks on sõnumite vastuvõtjal muutujad *SUTclosed* ning *SUTfull*, mis väljendavad üleminekuid hõivatud ning suletud seisunditesse.

Sõnumite vastuvõtja (*SUT*) automaadi mall on kujutatud joonisel 5.



Joonis 5 Sõnumite vastuvõtja (SUT-i) mudel

Sõnumite vastuvõtjal on neli seisundit:

- **Ootel** (*idle*) seisund on mudeli algseisund ning sõnumite vastuvõtja on selles seisundis ajal, mil ta on valmis sõnumite vastuvõtmiseks uute kilpidega ühenduma. Sõnumite vastuvõtjaga saab maksimaalselt olla ühenduses 1000 kilpi ning juba loodud ühenduste arvu hoitakse muutujas *noOfConnections*. Kui sõnumite vastuvõtja mudeliga on ühenduse loonud juba 1000 kilpi (*noOfConnections* > *max*), läheb sõnumite vastuvõtja hõivatud (*full*) seisundisse ning rohkem ühendusi ei saa kilbid luua seni, kuni ükski juba loodud ühendustest ei ole lõpetatud. Kui andmete edastus lõpetatakse läbi kanalite *i_send* ning *o_response*, läheb sõnumite vastuvõtja tagasi ootel seisundisse ning

ühenduse sulgemisel vähendatakse muutuja *noOfConnections* väärtust ühe võrra. Kui sõnumite vastuvõtjal ilmneb ootel seisundi vältel probleem, läheb see suletud seisundisse ning tõeväärtustüüpi muutuja *SUTclosed* muudetakse tõeseks (*SUTclosed = true*).

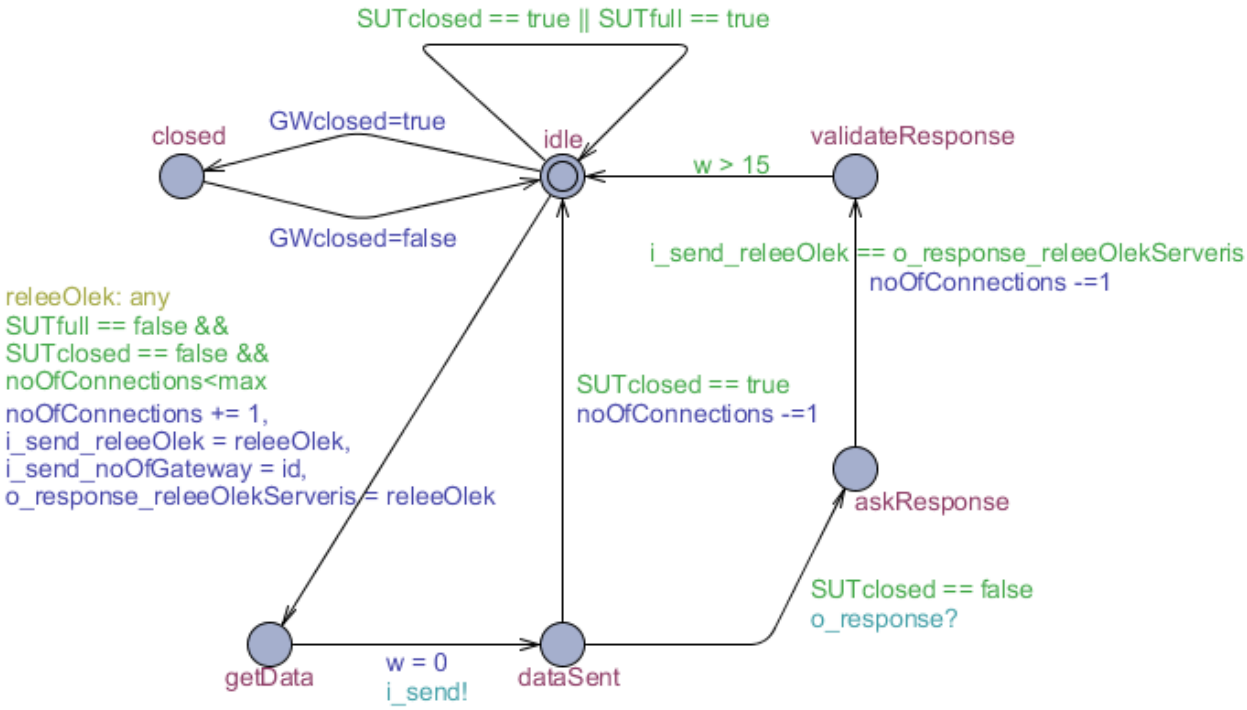
- **Andmete vastuvõtmise** (*data*) seisundisse läheb sõnumite vastuvõtja ajaks kui kilp või kilbid on sellega ühenduse loonud. Kui kilp ühendub sõnumite vastuvõtjaga läbi kanali *i_send* ning maksimaalne lubatud ühenduste arv ei ole täis (*noOfConnections* < *max*), läheb süsteem andmete vastuvõtmise (*data*) seisundisse. Kui kilbi poolt on andmed edastatud, küsitakse serverrakenduselt viimane teadaolev kilbi olek läbi sünkroniseerimiskanali *o_response* ning selle põhjal kontrollitakse, kas sõnumite vastuvõtjale jõudsid andmed korrektselt kohale. Seejärel läheb sõnumite vastuvõtja tagasi ootel seisundisse.
- **Hõivatud** (*full*) seisundis on sõnumite vastuvõtja ajal, kui sellega on ühenduse loonud juba 1000 kilpi (*noOfConnections* >= *max*). See tähendab, et rohkem kilpe antud ajahetkel sõnumite vastuvõtjaga ühendust luua ei saa ning tõeväärtustüüpi muutuja *SUTfull* muudetakse tõeseks (*SUTfull = true*). Selleks, et kilbid saaksid sõnumite vastuvõtjaga uuesti ühendusi luua, peab juba loodud ühenduste arv langema alla 1000. Hõivatud seisundist läheb sõnumite vastuvõtja tagasi ootel seisundisse siis, kui loodud ühenduste arv saab väiksemaks kui 1000 ning seejärel muudetakse muutuja *SUTfull* vääraks (*SUTfull = false*). Hõivatud seisundist on sõnumite vastuvõtjal võimalik minna ka suletud seisundisse juhul kui sõnumite vastuvõtja muutub uutele ühendustele kättesaamatuks. Hõivatud olekusse on sõnumite vastuvõtjal võimalik minna vaid juhul kui kilpide koguarv on vähemalt sama suur kui kilbi eksemplaride arv.
- **Suletud** (*closed*) seisundis on sõnumite vastuvõtja ajal, mil see on hoolustöödel või tehniliste probleemide tõttu kättesaamatu kõigile kilpidele. Suletud seisundi korral ei saa ükski kilp sõnumite vastuvõtjaga uusi ühendusi luua. Kui sõnumite vastuvõtja läheb suletud seisundisse, muudetakse tõeväärtustüüpi muutuja *SUTclosed* väärtus tõeseks (*SUTclosed = true*). Suletud seisundist on võimalik minna mudeli algseisundisse ning siis saavad kilbid uuesti sõnumite vastuvõtjaga ühendusi hakata looma. Lisaks on suletud seisundist võimalik minna hõivatud olekusse juhul kui kõiki ühendusi ei jõutud katkestada või sõnumite vastuvõtja oli mõne muu tehnilise probleemi tõttu ajutiselt suletud

seisundis. Ühtlasi muudetakse suletud seisundist ootel või hõivatud seisundisse minnes tõeväärtustüüpi muutuja *SUTclosed* vääraks (*SUTclosed* = *false*).

3.3.2 Kilbi mudel

Kilbi automaadi protsessid defineeritakse kilbi mudeli mallina eksemplaridena. See tähendab, et kuna kõik kilbid on sisult ja käitumiselt identsed, kuid erinevad üksteisest nime (*Gateway*[1], *Gateway*[2],...) ning sünkroniseerimiskanalitega kaasa antud identifikaatori väärtuse poolest. Kilpide eksemplarid on sõnumite vastuvõtja protsessiga üks-ühele seoses läbi sünkroniseerimiskanalite. Mudelite kirjeldamist mallina on kirjeldatud käesoleva magistritöö peatükis 3.3.3.

Kilbi (*Gateway*) automaadi mall on kujutatud joonisel 6.



Joonis 6 Kilbi mudel

Kilbil on kuus seisundit:

- **Ootel** (*idle*) seisund on mudeli algseisund. Kui kilp on ootel seisundis, proovib see sõnumite vastuvõtjaga ühendust luua. Kui sõnumite vastuvõtjaga ühendumine ei õnnestu, sest serverrakendus on suletud seisundis või maksimaalne lubatud ühenduste arv on täis (*SUTclosed == true || SUTfull == true*), jääb kilp ootel seisundisse ning proovib uuesti ühendust luua seni, kuni

see õnnestub. Kui ühenduse loomine õnnestub, läheb kilp seadistamise (*setId*) seisundisse..

- **Andmete seadistamise** (*getData*) seisund on mudeli seisund, mis on oluline muutujate väärtustamiseks. Ootel seisundist andmete seadistamise olekusse minekul väärtustatakse sõnumite vastuvõtjale edastatavad muutujad. Muutuja *i_send_noOfGateway* väärtustatakse kilbi identifikaatoriga. Ühtlasi genereeritakse juhuslik arv, mis emuleerib kilbi olekut. Juhusliku arvu genereerimist on kirjeldatud käesoleva magistritöö peatükis 3.3.4. Genereeritud juhusliku arvuga väärtustatakse muutujad *i_send_releeOlek* ja *o_response_releeOlekServeris*. Süsteem saab andmete seadistamise seisundisse minna vaid juhul, kui sõnumite vastuvõtja ei ole suletud ega hõivatud seisundites (*SUTclosed == false && SUTfull == false*).
- **Andmete saatmise** (*dataSent*) seisundisse läheb kilp ootel ning andmete seadistamise seisunditest juhul kui ühenduse loomine ning andmete edastus kilbi ja serverrakenduse vahel õnnestus. Sünkroniseerimiskanali *i_send* ilmnemisel saadetakse andmed sõnumite vastuvõtjale. Kui peale andmete vastuvõtmist läheb sõnumite vastuvõtja suletud seisundisse (*SUTclosed = true*), siis läheb kilp tagasi algseisundisse ehk ootel seisundisse ning proovib uuesti ühendust luua ning andmeid edastada. Kuna kilp edastab sõnumite vastuvõtjale andmeid 15-minutilise intervalliga, siis andmete saatmise seisundisse minnes nullitakse kell ning hakatakse lugema aega sellest hetkest alates.
- **Andmete küsimise** (*askResponse*) seisundisse läheb kilp juhul, kui saabub sünkroniseerimissignaal *o_response*. Selle tulemusena küsitakse sõnumite vastuvõtjalt viimane teadaolev kilbi olek, et kontrollida, kas andmed jõudsid sõnumite vastuvõtjale korrektselt kohale. Andmeid saab sõnumite vastuvõtjalt küsida vaid juhul kui sõnumite vastuvõtja pole suletud olekus (*SUTclosed == false*).
- **Andmete valideerimise** (*validateResponse*) seisund on mudelil selleks, et kontrollida, kas oodatud kilbi olek (*i_send_releeOlek*) ühtib sõnumite vastuvõtjale teadaoleva kilbi olekuga (*o_response_releeOlekServeris*). Kui oodatud tulemus ei vasta tegelikule tulemusele, lõpetab testimine ebaõnnestunud tulemusega. Lisaks on seisund selleks, et tagada 15-minutiline ühenduse loomise ning andmete edastuse intervall, sest kilbid edastavad sõnumite vastuvõtjale

infot iga 15-minutilise intervalli järel. Seega on andmete valideerimise ning ootel seisundeid ühendaval siirdel valvuri tingimus (*w>15*) ehk siiret on võimalik läbida minimaalselt 15 ajaühiku pärast peale andmete edastust.

- **Suletud** (c*losed*) seisundis on kilp ajal, kui see on hooldustöödel või ilmnevad tehnilised probleemid andmete edastamiseks. Kui kilp on suletud seisundis ehk tõeväärtustüüpi muutuja *GWclosed* on tõene (*GWclosed == true*), siis sõnumite vastuvõtjaga ühenduse loomist ega andmete edastamist ei toimu. Ühtlasi ei ole piiritletud aeg, kui kaua võib kilp olla suletud seisundis. Suletud seisundist saab kilp minna ainult mudeli algseisundisse ehk ootel seisundisse ning seejärel saab kilp uuesti hakata serverrakendusega ühendust looma ning andmeid edastama. Kui kilp läheb suletud seisundist ootel seisundisse, muudetakse muutuja *GWclosed* vääraks (*GWclosed == false*).

3.3.3 Kilbi mall

Tallinna tänavavalgustuse süsteemis on kilpide koguarv küllaltki suur ning nende arv võib aeg-ajalt muutuda. Seega on UPPAAL-i modelleerimis- ja mudelkontrolli keskkonnas võimalus defineerida mudeleid mallina, et vähendada mudelite kopeerimist ning testitava süsteemi hostide arvu muutumisel mudelite täiendamisele kuluvat aega. Lisaks võimaldab kilpide loomine mallina ilma mudelites suuri muudatusi tegemata testida süsteemi käitumist erineva kilpide arvu korral.

Kilpide malli defineerimiseks määratakse esmalt konstant, mis on täisarvu tüüpi ning millega määratakse kilpide koguarv. Kui kilpide koguarv muutub, muudetakse vaid vastava konstandi väärtust.

```
//Total number of gateways
const int totalNoOfGateways = 600;
```

Lisaks deklareeritakse uus massiivi tüüpi andmestruktuur. Deklaratsioonis määratakse massiivi nimi ning suurus. Suurus on antud parameetriliselt konstandiga *totalNoOfGateways*.

```
typedef int[1,totalNoOfGateways] gatewayId;
```

Kilbi mudeli eksemplarile määratakse parameeter, mille väärtus antakse kaasa ka sünkroniseerimiskanalis ning selle kaudu identifitseeritakse milliselt kilbilt andmeid edastatakse. Parameetri defineerimisel määratakse tüüp, mis antud juhul on *gatewayId* ning parameetri nimetuses, milleks on *id*.

```
const gatewayId id
```

Malli kasutamisel piisab kõikide kilpide defineerimiseks süsteemi deklaratsioonide all vaid malli kirjeldamisest. Soovitud arv kilpide eksemplare luuakse mudelite käivitamisel tulenevalt konstandist *totalNoOfGateways*.

```
system Gateway;
```

3.3.4 Mudelites juhusliku arvu genereerimine

Kilp edastab serverrakendusele oma oleku globaalse muutujaga *i_send_noOfGateway* ning kilbi olek on täisarvu tüüpi määramispiirkonnaga {0,1}. Mudelis kilbi oleku väärtustamiseks on kaks varianti.

Esimene variant kilbi oleku määramiseks on kilbi olek mudelitesse sisse kirjutada konstandina. Kuna kilbi mudelid vastavad kõik ühele mallile, siis konstandiga kilbi oleku määramisel saaksid kõik kilbid alati sama oleku ning see ei saa testimise jooksul muutuda. Sellisel juhul saab määrata konstandi, läbida testid ning muuta seejärel konstandi väärtust ja käivitada testid uuesti. Sellisel juhul jääksid testimata olukorrad, kui kilpidel on erinevad olekud ning kui kilbi olek testimise jooksul muutub.

Teine variant on genereerida kilbi olekud juhusliku arvu genereerimise teel, mida UPPAAL-i modelleerimiskeel toetab. See tähendab, et erinevad kilbid võivad saada erineva oleku ning kilbi olek võib testimise jooksul korduvalt muutuda.

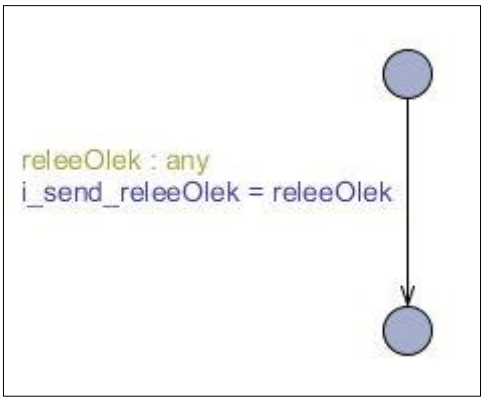
Juhusliku arvu genereerimiseks määratakse deklaratsioonide all konstant ehk lubatud maksimaalne väärtus, mida muutujale saab omistada. Lisaks deklareeritakse algväärtustatud muutuja, millele juhuslik väärtus peale genereerimist omistatakse.

```
//Maximum allowed random number
const int N = 1;

int i_send_releeOlek = 0;
```

Lisaks deklareeritakse uus massiivi tüüpi andmestruktuur. Kilbi olek peab olema täisarvtüüpi. Deklaratsioonis määratakse massiivi nimi ning massiivi pikkus on määratud eelnevalt defineeritud konstandiga N.

```
typedef int[0,N] any;
```

Juhuslik arv genereeritakse mudelis siirde mitte-deterministliku omistamise (*Select*) väljal ning kilbi oleku muutuja väärtustatakse sama siirde omistamise (*Update*) väljal. Juhusliku arvu genereerimine mudelitel on kujutatud joonisel 7.



Joonis 7 Juhusliku arvu genereerimine

3.3.5 Mudelite deklaratsioonid

UPPAAL-i modelleerimis- ja mudelkontrolli keskkonna mudelite koostamise vaates defineeritakse ka muutujad ning konstandid, mida mudelite kirjeldamisel kasutatakse.

Globaalsed deklaratsioonid

Globaalselt deklareeritakse mudelite muutujad, kellad, sünkroniseerimiskanalid ja konstandid, mida kasutatakse mitmes mudeli protsessis. [9]

Sõnumite vastuvõtja ja kilbi mudelitel on kolm globaalset konstanti, millest esimesega (*totalNoOfGateways*) määratakse kilpide koguarv. Sellest tulenevalt on kilpide koguarvu võimalik muuta ning jälgida süsteemi käitumist erineva kilpide koguarvu korral. Teise globaalse muutujaga (*max*) väljendatakse, mitmel kilbil on maksimaalselt

võimalik sõnumite vastuvõtjaga samaaegselt ühenduses olla. Kolmanda globaalse konstandiga määratakse maksimaalne arv, mis on võimalik juhusliku arvu genereerimise teel saada. Juhusliku arvuga genereeritakse kilbi olek. Globaalsete deklaratsioonide all kirjeldatakse ka muutujad, millest esimese (*noOfConnections*) väärtus näitab mitu kilpi on antud ajahetkel sõnumite vastuvõtjaga ühenduses. Kui globaalselt deklareeritud muutuja nime algus ühtib sünkroniseerimiskanali nimega, edastatakse selle muutuja väärtus testiadapterile või loetakse adapterist mudelisse. Antud testimisülesande korral on globaalsete deklaratsioonide all kirjeldatud muutuja, millega sünkroniseerimiskanalis edastatakse kilbi identifikaator (*i_send_noOfGateway*), muutuja, millega edastatakse kilbi olek (*i_send_releeOlek*) ning muutuja, millega edastab serverrakendus kilbi väärtuse (*o_response_releeOlekServeris*). Lisaks kirjeldatakse globaalsete deklaratsioonide all tõeväärtustüüpi muutuja *SUTclosed*, mis näitab, kas sõnumite vastuvõtja on suletud seisundis ning tõeväärtustüüpi muutuja *SUTfull*, mis näitab, kas sõnumite vastuvõtja maksimaalne lubatud ühenduste arv on täis. Algselt on tõeväärtustüüpi muutujad väärad (*false*).

```
//Total number of gateways
const int totalNoOfGateways = 600;

//Maximum number of allowed connections
const int max = 1000;

//Maximum allowed random number
const int N = 1;

int noOfConnections = 0;
int i_send_noOfGateway = 0;
int i_send_releeOlek = 0;
int o_response_releeOlekserveris = 0;

typedef int [1, totalNoOfGateways] gatewayId;
typedef int [1, N] any;

bool SUTclosed = false;
bool SUTfull = false;
```

Globaalsete muutujate juures defineeritakse ka sünkroniseerimiskanalid, mille kaudu mudelite protsessid omavahel on sünkroniseeritud. Sisendkanalid ehk kanalid, millega seostatakse andmeid mudelitest testitavasse süsteemi, on tähistatud eesliitega „i_“ ning väljundkanalid, millega seostatakse andmed testitavast süsteemist mudelitele, on tähistatud eesliitega „o_“.

```
// synchronisation channels
chan i_send, o_response;
```

Lokaalsed kilbi muutujad

Kilbi mudelimallile defineeritakse lokaalsed muutujad, kellad ja konstandid, mida kasutatakse vaid selle malli protsessides.

Kilbi mudelimalli juures defineeritakse kell (*w*), mis arvestab aega, millal kilp saab hakata uuesti serverrakendusega ühendust looma. Kilp võtab sõnumite vastuvõtjaga ühendust iga 15-minutilise intervalli järel. Lisaks defineeritakse lokaalselt kilbi mudeli juures tõeväärtustüüpi muutuja *GWclosed*, mis näitab kas kilbi uks on suletud. Muutuja *GWclosed* on mudelite käivitamisel algväärtustatud väärtusega *false*.

```
clock w;
bool GWclosed = false;
```

Süsteemi deklaratsioonid

Süsteemi deklaratsioonide all defineeritakse kilbi malli (*Gateway*) ning sõnumite vastuvõtja (*SUT*) malli protsessid, mis malli parameetrite initsialiseerimise tulemusena moodustavad konkreetse mudeli.

```
// Processes to be composed into a system
system Gateway, SUT;
```

3.3.6 Mudelitele seatud nõuded

Kokku koosnevad mudelid sõnumite vastuvõtja ning kilpide protsessidest. Kilbi malli kõik protsessid käituvad küll sisult identselt, kuid nõuete täidetavust kontrollitakse kõigi protsesside paralleelkompositsiooni korral, et avastada modelleerimisel tehtud vigasid.

Sõnumite vastuvõtja (SUT-i) ja kilbi mudelid peavad vastama alljärgnevatele nõuetele:

Saavutatavuse nõuded

Saavutatavuse nõuetega väljendatakse, et nii SUT-i kui kilbi käitumises peab olema võimalik jõuda kõigi mudelites kirjeldatud seisunditeni.

Viimasena kirjeldatud nõue peab täidetud olema vaid juhul kui kilpide koguarv on suurem kui maksimaalne lubatud ühenduste arv. Kui kilpide koguarv ei ole suurem kui

maksimaalne lubatud ühenduste arv, siis sõnumite vastuvõtjal ei ole võimalik hõivatud (*full*) seisundisse minna, seega ei saa ka antud nõue täidetud olla.

```
E<> SUT.idle
E<> SUT.data
E<> SUT.closed
E<> forall (e : gatewayId) Gateway(e).idle
E<> forall (e : gatewayId) Gateway(e).getData
E<> forall (e : gatewayId) Gateway(e).dataSent
E<> forall (e : gatewayId) Gateway(e). askResponse
E<> forall (e : gatewayId) Gateway(e). validateResponse
E<> forall (e : gatewayId) Gateway(e).closed

E<> SUT.full
```

Ohutusomadused

Ohutusomadustega väljendatakse tingimusi, mida täitmise käigus ei tohi rikkuda. Näiteks ei tohiks mudelites olla tupikuid, mida väljendab *not deadlock* nõue. Teine kirjeldatud nõue väljendab, et mudelites ei tohiks olla olukorda, kus SUT on suletud seisundis, kuid kilpidel siiski õnnestub SUT-iga ühendust luua. Kolmas kirjeldatud nõue on analoogiline teisega, et kui SUT on hõivatud seisundis, siis ei tohi uutel kilpidel olla võimalik SUT-iga ühendust hakata looma.

```
A[] not deadlock
E[] not (SUT.closed and forall (e : gatewayId) Gateway(e).getData)
E[] not (SUT.full and forall (e : gatewayId) Gateway(e).getData)
```

Seda, kas koostatud mudelid vastavad seatud nõuetele, saab kontrollida UPPAAL-i mudeli kontrollijaga. Nõuete kontrolli tulemusi kuvatakse iga kirjeldatud nõude rea lõpul ning nõuete staatuse väljal. Nõuete staatuse väljal kuvatakse iga nõude kohta roheliselt märge, kui koostatud mudelid vastavad seatud nõuetele. Kui nõude kohta kuvatakse punane märge, siis koostatud mudelid kirjeldatud nõuetele ei vasta.

Nõuete kontrolli tulemused on Lisades 1 ja 2. Lisas 1 on nõuete kontrolli tulemused juhul kui süsteemis olevate kilpide koguarv on väiksem kui maksimaalne lubatud ühenduste arv. Kui kilpide koguarv on väiksem kui maksimaalne lubatud ühenduste arv, siis ei ole sõnumite vastuvõtjal võimalik hõivatud (*full*) seisundisse jõuda. Lisas 2 on nõuete kontrolli tulemused juhul, kui süsteemis olevate kilpide koguarv on suurem kui maksimaalne lubatud ühenduste arv ning mudelid peavad vastama kõigile kirjeldatud nõuetele.

3.4 DTRON

3.4.1 Adapter

DTRON-i adapter on kirjutatud java programmeerimiskeeles. Adapter teisendab sünkroniseerimiskanalites edastatava info mõlemas suunas - nii mudelitelt testitavale süsteemile kui vastupidi suunas sobivale kujule. Iga sünkroniseerimiskanali kohta kirjeldatakse eraldi adapter. Tallinna tänavavalgustuse süsteemi kõik kilbi mudeli protsessid kasutavad ühte sisend sünkroniseerimiskanalit, seega piisab ühe adapteri kirjeldamisest.

UPPAAL TRON-i kasutamisel on adapterite programmeerimine küllaltki keeruline, sest adapteris tuleb ühtlasi kirjeldada testitava süsteemi ning mudelite vahel liikuvate andmete spetsiifikat. TRON-i edasiarenduse DTRON-i kasutamisel üks eelis seisneb selles, et DTRON-is tehakse sisemiselt suur osa andmeteisenduse kirjeldamisest ära. Seega võtab DTRON-i adapterite programmeerimine vähem aega ning suure kanalite arvu korral ei ole adapterite kirjeldamine ning nende täiendamine liiga suur töö.

Adapteris luuakse esmalt DTRON-iga ühendus.

```
//connect dtron
Dtron dtron = new Dtron();
dtron.connect();
```

Iga UPPAAL-i kanali jaoks luuakse adapter, kus kirjeldatakse sisend- ning väljundkanalite nimed ning kanalites edastatavad andmed.

Kõik kilbid kasutavad andmete edastamiseks sama kanalit. Kanalid kirjeldatakse adapteris ilma eesliiteta „i_“ alljärgneval kujul:

```
//listen to channel
IDtronChannel sendDataChannel = new DtronChannel("send");
```

Sünkroniseerimiskanalis edastatakse ka kõigi globaalsete muutujate väärtused, mille nimetuse eesliide ühtib sünkroniseerimiskanali nimega. Seega kui sünkroniseerimiskanali nimi on *i_send*, siis sünkroniseerimiskanalis edastatakse ka muutuja *i_send_noOfGateway* väärtus. Adapteris selle muutuja väärtuse teadasaamiseks eemaldatakse sünkroniseerimiskanali nimele viitav eesliide. Muutuja *noOfGateway*

väärtus, milles hoitakse sõnumite vastuvõtja poole pöörduva kilbi identifikaatorit, küsitakse adapteris mudelilt alljärgneval kujul:

```
gwNo = connectChanValue.getVariables().get("noOfGateway");
```

Seejärel pannakse adapteris kokku sõnum, mis saadetakse serverrakendusele. Sõnumi struktuur on kirjeldatud käesoleva magistritöö peatükis 3.1.

```
bytes[0] = (byte)gwID;

gwNo = connectChanValue.getVariables().get("noOfGateway");
bytes[1] = (byte)gwNo;

bytes[2] = (byte)releeID;

releeOlek = connectChanValue.getVariables().get("releeOlek");
bytes[3] = (byte)releeOlek;

bytes[4] = (byte)end;

//Send data to server
for (int i = 0 ; i<=parameetriteArv; i++){
      System.out.println("Byte array value:" + (int)bytes[i]);
      serverSimulator.sendData(bytes[i]);
}
```

Kilbilt andmete edastamine serverrakendusele on ühepoolne ehk kilp edastab andmed serverrakendusele, kuid serverrakendus ei saada kinnitust andmete kohalejõudmise kohta. Samuti ei saada serverrakendus infot, kui sõnum ei jõudnud kohale. Selleks, et mudelites saaks hinnata, kas andmed jõudsid serverrakendusele korrektselt kohale, küsib kilp serverrakenduselt konkreetse kilbi viimase serverrakendusele teadaoleva oleku.

```
IDtronChannel sendDataChannel = new DtronChannel("response");

serverSimulator.getGatewaySate(gwNo);
responseValue = getReleeValue(3);
```

Serverrakenduselt saadud andmete tulemusena on võimalik hinnata, kas kilbi poolt serverrakendusele saadetud ning serverile teadaolev viimane kilbi olek ühtivad. Adapter edastab serverrakenduselt saadud kilbi oleku läbi Spread serveri mudelile ning mudelis kontrollitakse väärtuste ühtivust. Andmed saadakse mudelilt adapterile ning edastatakse adapterilt mudelile alati Map<String, Integer> paaridena.

```
data.put(responseVariable, responseValue);
IDtronChannelValued valued = sendDataChannel.constructValued(data);

try {
	// log outgoing channel
	System.out.println("Sending channel value: " + valued);
	getDtron().send(valued);
} catch (SpreadException e) {
	e.printStackTrace();}
```

Kui kõik testid on läbitud, jääb adapter ootama ENTER klahvi vajutamist, mille tulemusena suletakse DTRON-i ühendus.

```
//disconnect dtron (ENTER terminates)
System.in.read();
dtron.disconnect();
```

Koostatud adapteri kood on lisas 3.

3.4.2 Sõnumite vastuvõtja simulaator

Tallinna tänavavalgustuse süsteemi testkeskkond ei ole kogu aeg kättesaadav. Seega on töö käigus tehtud vähendatud funktsionaalsusega serverrakenduse simulaator, millega adapter loob ühenduse. Sõnumite vastuvõtja simulaator on kirjutatud java programmeerimiskeeles. Sõnumite vastuvõtja simulaatori funktsionaalsus:

- Simulaator võtab kilpidelt saabuvaid sõnumeid vastu. Kilbid saadavad sõnumite vastuvõtjale infot baitide jadana. Simulaator võtab kilpidelt saabuvad andmed vastu.
- Peale andmete saabumist, salvestatakse üle konkreetsele kilbi olek. Seega hoiab simulaator iga kilbi kohta ainult selle viimast olekut. Olekute muutuste ajalugu ei säilitata.
- Kui simulaatorilt küsitakse konkreetse kilbi olekut, siis edastab see baitide jadana sõnumi konkreetse kilbi viimase salvestatud oleku kohta.

3.4.3 Testide käivitamine

Kui mudelid ning adapter on koostatud, tuleb testide käivitamiseks esmalt käivitada käsurealt Spread server, milleks tuleb koostada konfiguratsioonifail, kus määratakse IP

aadress koos pordi numbriga. Lisaks on konfiguratsioonifailis (*spread.conf*) võimalik määrata erinevaid lisatingimusi ehk näiteks seda, kui detailselt toimub logimine.

Spread.conf

```
Spread_Segment  127.0.0.255:4803 {
      localhost         127.0.0.1
}
```

Seejärel käivitatakse programmeeritud adapterid ning viimasena DTRON ehk käsurearakendus, millele antakse ette koostatud mudelite fail ning täpsustatakse vajadusel täiendavalt tingimusi.

```
java -jar dtron.jar -f <FILE> [-l <MSEC> | -P <VALUE> |
                     -v <LEVEL>] -o <UNITS> [-s <HOST>] -u <MSEC>
```

-f,--fail	<FILE>	UPPAAL-iga koostatud mudeli XML-fail
-l,--latentsus	<MSEC>	Sisendite lubatud hilistumine millisekundites
-o,--aegumine	<UNITS>	TRON-i aegumise ajaühik
-P,--viivitus	<VALUE>	Viivitus siiretel väle (*eager*): viivitus siirdel on nii lühike kui võimalik aeglane (*lazy*): viivitus siirdel on nii pikk kui võimalik juhuslik (*random*): viivituse piirid on defineeritud mudelis (vaikimisi)
-s,--spread	<HOST>	Spread serveri hosti aadress koos pordi numbriga (vaikimisi: *localhost:0*)
-u,--ajaühik	<MSEC>	TRON-i ajaühik millisekundites
-v,--detailsus	<LEVEL>	Testide logimise spetsiifilisuse tase (vaikimisi: 9) [22]

DTRON-i käivitamisel on kohustuslik määrata –f <FILE>, -o <UNITS> ja –u <MSEC>. Ülejäänud parameetrid võib jätta sisestamata ning sellisel juhul kasutatakse parameetrite vaikeväärtusi. Koostatud mudelite XML-fail peab asuma DTRON-iga samas kaustas.

Testid käivitatakse käsurealt:

```
java -jar dtron.jar -f streetlight.xml -o 1000 -u 4000 –P eager
```

3.4.4 Testimise tulemus

Testi väljundis kuvatakse informatsioon kõigi edastatud sünkroniseerimiskanalite ning sünkroniseerimiskanalis edastatud globaalsete muutujate väärtuste kohta. See aitab peale testi läbimist testijal selgemalt hinnata, millised olukorrad on läbitud. Adapteri programmeerimisel tuleks mudelitest adapterile saabuvad ning adapterist mudelitele tagasi saadetavad väärtused logida, et vajadusel peale testimise lõppu oleks selgem ülevaade testandmete kohta.

Testide käivitamisel on muudetud mudeleid ning kilbi eksemplaride koguarvuks on määratud 6 kilpi, sest kilbi eksemplaride arvu suurendamisel ilmnesid DTRON-is mäluprobleemid. Töö käigus on läbitud teste juhul kui kilpide koguarv on suurem kui maksimaalne võimalik ühenduste arv ning ühtlasi on läbitud teste kui kilbi eksemplaride koguarv on väiksem kui sõnumite vastuvõtja lubatud maksimaalne ühenduste arv. Mõlemal juhul testid õnnestusid ning detailne testimise väljund on lisas 4.

Selleks, et kontrollida, kas mudelid ning adapterid töötavad korrektselt ning vigade ilmnemise korral testi läbimine ebaõnnestub, muudeti ajutiselt sõnumite vastuvõtja simulaatorit. Simulaatorit muudeti nii, et see tagastaks kilbi oleku väärtuse, mis on väljaspool kilbi juhusliku arvu genereerimise piire ning mis mudelis ei saa kunagi tekkida. Seega muudeti serveri simulaatorit nii, et see tagastaks alati kilbi olekuks 100.

```
//Testing value
int testimiseksReleeOlek = 100;

//For testing
serverBytes[3] = (byte)testimiseksReleeOlek;
```

Kui sõnumite vastuvõtja simulaatorit muudeti nii, et see tagastaks serveris olevaks kilbi olekuks alati vale väärtuse, siis testide läbimine ebaõnnestus. Testi ebaõnnestumise põhjuseks oli, et testi tegelik väljund ei vastanud oodatud väljundile. Lisaks kuvati DTRON-i väljundis detailsemat infot muutujate ning nende väärtuste kohta, et testijal oleks vea põhjust kergem tuvastada.

```
TEST FAILED: Observed output has wrong variable value(s).
```

Ebaõnnestunud testi väljund kui serveri simulaator tagastab alati sobimatu kilbi oleku, on lisas 5.

4 Modelleerimislahenduste võrdlev analüüs

Tallinna Tänavavalgustuse süsteemis on väga suur hulk hoste, mis muudab testimise üheks keeruliseimaks etapiks skaleeruva testimudeli koostamise. Leidmaks parimat hajussüsteemide modelleerimise varianti, koostati töö käigus alternatiivsed mudelite lahendused.

4.1 Parametriseeritud mallid ja sünkroniseerimiskanalid

UPPAAL-i modelleerimiskeel võimaldab defineerida mudelite parametriseeritud malle ning sünkroniseerimiskanaleid. Tänu sellele on võimalik identselt käituvate hostide käitumisloogikat kirjeldada ühe mudeli mallina. Sama malli protsesside sünkroniseerimiseks on otstarbekas kirjeldada sünkroniseerimistingimusi kanalite massiivina ning indekseerida kahe protsessi vahelisi interaktsioone kasutades vastavate protsesside indekseid. Niimoodi parametriseeritud sünkroniseerimiskanalid võimaldavad lihtsustada adapterite programmeerimist, sest adapter koostatakse iga sünkroniseerimiskanali kohta eraldi. Kui sünkroniseerimiskanalid on koostatud massiivina, piisab ühe adapteri programmeerimisest.

Tulenevalt UPPAAL TRON-i piirangutest võrreldes UPPAAL-i automaatide üldise kirjelduskeelega, ei aktsepteeri TRON ülal kirjeldatud kanalimassiive. UPPAAL TRON on ehitatud UPPAAL-i modelleerimiskeele ühe varasema versiooni baasil, mil parametriseeritud sünkroniseerimiskanalid ei olnud veel toetatud. Seega ei saa TRON-i ja DTRON-i kasutades parametriseeritud sünkroniseerimiskanaleid mudelite modelleerimisel kasutada. Küll aga lihtsustab kanalimassiivide kasutamine mudeleid nende verifitseerimisel UPPAAL-is.

Parametriseeritud mudelimalle ja kanalimassiive kasutavad mudelid on esitatud lisas 6.

4.1.1 Kanalimassiivide kasutamise eelised ja puudused

Kanalimassiivide kasutamise peamised eelised:

- Kanalimassiivide kasutamisel genereeritakse see massiiv deklaratsioonis antud parameetrite alusel automaatselt ning seega on adapterite programmeerimine ning hooldamine lihtsam ja vähem aeganõudev.
- Modelleerimine on kergem ning mudelid kergemini loetavad, sest ühte sünkroniseerimiskanalit saab kasutada erinevate mudeli eksemplaride korral. Ühtlasi ei ole sünkroniseerimiskanalitele nende identifitseerimiseks vaja kaasa anda täiendavad muutujaid, mis muudavad mudelite loetavuse keerulisemaks, sest tuleb jälgida, et muutujad väärtustatakse õigesti.

Kanalimassiivide kasutamise peamine puudus:

- Kui hajussüsteemis on palju erinevalt käituvaid hoste ning sünkroniseerimiskanalid on erinevate siirete vahel, siis parametriseeritud mallide ning kanalimassiivide kasutamine ei ole võimalik ning kõikide hostide käitumisloogika tuleb kirjeldada ükshaaval.

4.2 Individuaalsed kanalid protsesside vahel

Protsessidevaheliste individuaalsete kanalite spetsifitseerimise korral defineeritakse mudelites nii kilpide kui serverrakenduse eksemplarid kui neid ühendavad kanalinimed ükshaaval, mis on üldjuhul suurte süsteemide korral liiga keeruline ning aeganõudev. Tallinna tänavavalgustuse süsteemi mudelite kirjeldamist lihtsustab asjaolu, et hostid käituvad identselt. See tähendab, et identselt käituvate hostide modelleerimiseks saab määrata modelleerimise kitsendused ning kirjeldada vähem mudelite eksemplare, kui on reaalselt hajussüsteemi hoste.

Individuaalseid protsesside-vahelisi kanaleid kasutavad mudelid on esitatud lisas 7.

4.2.1 Modelleerimise kitsendused

Suurte hajussüsteemide korral tuleks individuaalsete protsesside-vaheliste kanalite defineerimiseks teha enne modelleerimist rida eeldusi, mis määravad missugustel tingimustel saab mudelist teha adekvaatseid järeldusi süsteemi kui terviku töö kohta. Testmudeli koostamisel seati piir testis osalevate kilpide arvule. Mudel koosneb 10 kilbi ning 10 sõnumite vastuvõtja protsessist, mis on sisult ja käitumiselt üksteisega identsed, kuid erinevad nime (*Gateway1, Gateway2,...*) ning sünkroniseerimiskanalite

järjekorranumbri poolest (*i_send1, i_send2,...*). Kilpide ning sõnumite vastuvõtjate protsessid on üks-ühele seoses läbi neid ühendavate unikaalsete sünkroniseerimiskanalite.

Modelleerimisel tehti mitmeid eeldusi, et modelleerimise ning mudelite hooldamise töömahtu vähendada:

- Tegeliku kilpide koguarvu asemel sisaldab mudel kilbi automaadi malli 10 eksemplari, mis esitavad antud testijuhu puhul süsteemi käitumist. Iga kilbi malli eksemplari juures määratakse unikaalsed sünkroniseerimiskanalid.
    - Kilbi malli eksemplarid (*Gateway1, Gateway2, Gateway3,...*)
    - Sünkroniseerimisekanalid (*i_send1, i_send2, i_send3,...*)
- Iga kilbi mudeli kohta luuakse paralleelselt üks sõnumite vastuvõtja ehk SUT-i automaadi eksemplar ehk luuakse 10 kilbi ning 10 sõnumite vastuvõtja (*SUT*) protsessi.
    - Sõnumite vastuvõtja mudeli eksemplarid (*SUT1, SUT2, SUT3,...*)
- Kilbi ning serverrakenduse protsessid moodustavad omavahel suhtlevate automaatide paari. Iga kilbi ning serverrakenduse protsessi paar kasutab sünkroniseerimiseks oma kanaleid.
    - Kilp ja serverrakendus, mille identifikaatorid on „1“, suhtlevad omavahel läbi sünkroniseerimiskanali, mille identifikaator on „1“.

4.2.2 Protsessidevaheliste individuaalsete kanalite eelised ja puudused

Protsessidevaheliste individuaalsete kanalite kasutamise peamine eelis modelleerimisel:

- Sõnumite vastuvõtja ja kilp suhtlevad läbi unikaalsete sünkroniseerimiskanalite, seega on välistatud olukorrad, et sünkroniseerimiskanalites edastatakse info valele saajale või sünkroniseerimiskanalis edastatav muutuja oleks vigaselt väärtustatud.

Protsessidevaheliste individuaalsete kanalite kasutamise peamised puudused mudelite modelleerimisel:

- Kui hajussüsteemis on suurem hulk erinevalt käituvaid hoste, on mudelite loomine ning süsteemile seatud nõuete muutumisel mudelite täiendamine

keeruline ning ajakulukas, sest protsesside arv mudelis on väga suur ja iga omavahel suhtleva protsesside paari vahelised kanalid tuleb eraldi defineerida.

- Suure kanalite arvu korral on adapterite programmeerimine väga aeganõudev, sest adapterite kood kasvab kanalite arvu kasvades. Süsteemile seatud nõuete muutumisel on adapterite täiendamine ning muutuvate nõuetega kaasajastamine liialt suur töö.
- Suurte süsteemide korral tehakse protsessidevaheliste individuaalkanalite kasutamisel modelleerimisel lisaeeldusi, et modelleerimise mahtu vähendada. Eelduste tegemisega võib tekkida oht, et kirjeldatud kitsendused ei ole sobilikud ning sellest tulenevalt koostatud mudelid ei käitu õigesti.
- Kui süsteemi modelleerimisel tehakse kitsendused ning modelleeritakse tegelikust arvust vähem süsteemi eksemplare, võimaldab DTRON testimiseks laadida mudeleid mitmesse masinasse paralleelselt. See tähendab, et sama mudelite faili saab paralleelselt käivitada mitmes käsurea aknas, et tekitada reaalsele serverile tegelikkusega sarnane koormus. Mitmes käsureaaknas samaaegselt mudelite käivitamine ning tulemuste jälgimine on keeruline ning vigade ilmnemisel võtab vea põhjuse avastamine rohkem aega.

Analüüsi järeldused

Magistritöö tulemusena leiti, et UPPAAL-i modelleerimis- ja mudelkontrollikeskkond on sobilik hajussüsteemide modelleerimiseks. UPPAAL-i modelleerimiskeele esitusvõimsus on paljudel juhtudel piisav, et lähtuvalt konkreetse testitava süsteemi spetsiifikast on võimalik leida sobilik modelleerimislahendus. Tallinna tänavavalgustuse süsteemi näite analüüsil selgus, et väiksemate süsteemide korral on võimalik kõikide hostide käitumine kirjeldada erinevate mudeli mallidena. Juhul kui süsteemis on suurem hulk identselt käituvaid hoste, on nende käitumist võimalik kirjeldada üldisema parametriseeritud mallina.

Mudelite konstrueerimine UPPAAL-is võib muutuda liialt keeruliseks väga suurte süsteemide korral kui süsteemis on suur hulk erinevalt käituvaid hoste, mis nõuab erinevate mudelimallide ja paljude paralleelsete sünkroniseeritud protsesside defineerimist. Sellisel juhul on keeruline tagada mudelite õigsus ning nende koostamine ja hooldamine võtab väga palju aega.

Testide genereerimise ja täitmise vahendi UPPAAL TRON-i kasutamisel on mitmeid puudusi. TRON üksi ei sobi hajussüsteemide mudelipõhiseks testimiseks ning lisaks põhineb see UPPAAL-i modelleerimiskeele piiratud versioonil. See tähendab, et juba uuemates UPPAAL-i modelleerimiskeele versioonides toetatud funktsionaalsust ei ole TRON-is võimalik kasutada. Hajustestimiseks on vaja täiendavalt kasutada testimisvahendit DTRON.

Lisaks leiti töö tulemusena, et hajussüsteemide mudelipõhiseks testimiseks loodud DTRON-iga on adapterite programmeerimine lihtne väiksemate süsteemide korra ning juhul kui süsteemis ei ole suur arv sünkroniseerimiskanaleid. Vastasel juhul võib adapterite programmeerimine ning kaasajastamine muutuda liiga keeruliseks ning sellega kaasneb suur ajakulu. Suure sünkroniseerimiskanalite arvu korral on adapterite programmeerimine aeganõudev, sest adapteri keerukus kasvab sünkroniseerimiskanalite arvu kasvamisel. Selle tulemusena ei kiirenda mudelipõhine testimine testimisprotsessi ega taga paremat testitava süsteemi kvaliteeti.

DTRON-iga on võimalik andmeid mudelite ning adapteri vahel edastada vaid <String, Integer> paaridena, mis seab täiendavaid piirangud mudelite kujule. Kuna Tallinna Tänavavalgustuse süsteemis võib kilbi olekute arv kasvada ehk juhul kui sõnumite vastuvõtja on hõivatud või suletud, siis vahepeal edastamata jäänud olekud ei lähe kaduma, vaid need edastatakse järjestikku sõnumite vastuvõtjale. Kuna DTRON-iga on võimalik edastada andmeid kilbilt adapterile vaid kujul, kus globaalsele muutujale vastab vaid täisarvtüüpi väärtus, siis ei ole võimalik edastada kõiki olekuid, mis on kilbil vahepealsel ajal ilmnenud.

Täiendavaid edasiarendusvõimalusi on kolm. Esmalt tuleks tähelepanu pöörata mudelite käivitamisele vahendiga DTRON, sest DTRON võimaldab mudelite käivitamisel määrata täiendavaid lisatingimusi. Täpsemalt tasuks uurida, kuidas DTRON-i käivitamisel kaasa antavad parameetrid ja lisatingimused, sealhulgas hilistumised ning viivitused siiretel, mõjutavad testide tulemusi, läbitud testide hulka ning veaolukordade leidmise tõenäosust.

Testide käivitamisel kilbi eksemplaride arvu suurendamisel ilmnesid DTRON-iga skaleeruvusprobleemid ehk teste oli võimalik läbida vaid juhul kui kilbi eksemplaride koguarv jäi väikemaks kui 10. Tegelikkuses on Tallinna Tänavavalgustuse süsteemis umbes 600 kilpi, seega tuleks töö edasiarendusena uurida, milles täpselt seisnesid DTRON-i mäluprobleemid kilbi eksemplaride arvu suurendamisel.

Kolmanda edasiarendusena tuleks Tallinna Tänavavalgustuse süsteemi vaadelda laiemas funktsionaalsuses ning kirjeldada mudelid ka teiste süsteemiosade kohta. Sellest tulenevalt valmiksid kogu süsteemi kirjeldavad mudelid. Mudelid tasuks koostada nii, et saaks jälgida ja testida süsteemi kõiki osasid eraldiseisvalt ning ühtlasi testida kõigi süsteemiosade koostoimimist. Modelleerimisel tuleks jälgida, et mudelite keerukus ning maht ei kasvaks liiga suureks.

Kokkuvõte

Käesolevas töös on antud ülevaade mudelipõhisest testimisest kui automatiseeritud testimise alaliigist, mis aitab tagada testitava süsteemi paremat kvaliteeti. Mudelipõhise testimise peamine eelis klassikalise automatiseeritud testimise ees on testide automaatse genereerimise võimalus ja suurema testide hulga läbiproovimine. Testija ei pea programmeerima kõiki testlugusid kirjeldavaid teste, vaid mudelipõhise testimise korral kirjeldatakse mudelid, millest genereeritakse testlood automaatselt. Süsteemile seatud nõuete muutumisel peaks piisama mudeli ning adapterite muutmisest. Kuid liiga suure mudelite ning sünkroniseerimiskanalite arvu korral võib mudelite ning adapterite täiendamine olla liiga suur ajakulu ning ei taga soovitud efektiivsuse tõusu.

Hajussüsteemide mudelipõhisel testimisel on kaks peamist keerukust, milleks on juhitavuse ja jälgitavuse probleem. Juhitavuse ja jälgitavuse probleemid ilmnevad põhjusel, et hajussüsteemide korral on oluline jälgida hostidevahelise info liikumist ning tagada selle õige ajastus. Töös kasutatav hajussüsteemide mudelipõhise testimise vahend DTRON lähtub sellest, et testi juhitavus ja jälgitavus oleks tagatud.

Töös on lähemalt vaadeldud modelleerimis- ja mudelkontrollikeskkonda UPPAAL ning reaalajasüsteemide mudelipõhiseks musta-kasti testimiseks mõeldud testimisvahendit UPPAAL TRON. UPPAAL TRON genereerib lähtuvalt koostatud mudelitest testlood ning täidab need. Kuivõrd UPPAAL TRON ei sobi otseselt hajussüsteemide mudelipõhiseks testimiseks, on antud töös käsitletud ka DTRON-i, mis põhineb UPPAAL TRON-il, kuid on selle edasiarendus hajussüsteemide mudelipõhiseks testimiseks.

Töö teises pooles on koostatud Tallinna Tänavavalgustuse süsteemi näitel sõnumite vastuvõtjat ning kilpe kirjeldavad mudelid. Tallinna Tänavavalgustuse süsteem koosneb kesksest serverrakendusest ning üle Tallinna asuvatest kilpidest, mis edastavad kilpides toimuvat muudatuste infot perioodiliselt sõnumite vastuvõtjale. Töö käigus koostati kesksel serverrakendusel asuva sõnumite vastuvõtja ning üle Tallinna paiknevate kilpide mudelid.

Ühtlasi on modelleerimisel lähtutud asjaolust, et mudelite koostamine ning hiljem nende täiendamine oleks võimalikult lihtne ning vähe aega nõudev. Kilpide mudelid on loodud mallina ehk on kirjeldatud üks kilbi käitumist iseloomustav mudeli automaat ning soovitud arv kilbi eksemplare luuakse mudelite käivitamisel automaatselt. Selline lähenemine võimaldab mudelites suuri muudatusi tegemata testida süsteemi käitumist kilpide arvu muutumise korral. Serverrakenduse modelleerimisel on silmas peetud asjaolu, et tulenevalt tehnilistest piirangutest saab sõnumite vastuvõtjaga olla samaaegselt ühenduses maksimaalselt 1000 kilpi. Hetkel on süsteemiga seotud kilpide arv umbes 600, kuid sõnumite vastuvõtja on modelleeritud nii, et mudeli globaalsete parameetrite muutmisel saab testida süsteemi käitumist ka juhul kui kilpide koguarv kasvab suuremaks kui võimalik maksimaalne ühenduste arv.

Ühtlasi on välja töötatud alternatiivsed testitava süsteemi modelleerimislahendused, milleks on vastavalt parametriseeritud mallide ja sünkroniseerimiskanalite kasutamine ning teiselt poolt mitte-parametriseeritud kilbi ja hosti protsesside paarid, mille korral ei teki konkurentsi hostiga ühenduse loomisel. Kanalimassiivide lahendus ei sobi kasutamiseks, sest teste genereeriv UPPAAL TRON põhineb UPPAAL-i modelleerimiskeele varasemal versioonil, kus kanalimassiivid ei olnud veel lubatud. Teise alternatiivse lahendusena loodud protsesside vaheliste individuaalkanalite kirjeldamine on väga ajamahukas, sest koostatud mudeleid ning sünkroniseerimiskanaleid on väga palju. Lisaks on sellisel juhul testiadapterite loomine ning täiendamine aeganõudev, sest adapteri keerukus kasvab sünkroniseerimiskanalite arvu kasvamisel.

Kasutatud materjalid

[1] Markvardt, M. Sissejuhatus mudelipõhisesse testimisse. [WWW] http://193.40.251.102/tiki-download_wiki_attachment.php?attId=314 (14.12.2014)

[2] Markvardt, M. Tarkvara testimist käsitlev juhendmaterjal. [WWW] http://www.riso.ee/sites/default/files/Testimise_juhis.doc (20.01.2015)

[3] UPPAAL web page. [WWW] http://uppaal.org/ (10.02.2015)

[4] Zander, J., Schieferdecker, I., Mosterman, P.J. Model-Based Testing for Embedded Systems. United States of America : CRC Press, 2012.

[5] Hireons, R.M. Testing Distributed Systems. [WWW] http://antares.sip.ucm.es/tarot09/index_files/Hierons-TAROT09.pdf (21.01.2015)

[6] Larsen, K. G., Mikučionis, M., Nielsen, B. UPPAAL TRON User manual [WWW] http://people.cs.aau.dk/~marius/tron/manual.pdf (08.03.2015)

[7] Roo, R. Veebirakenduste mudelipõhine testimine asukohapõhise tarkvara näitel. [WWW] http://dspace.utlib.ee/dspace/bitstream/handle/10062/14755/roo_rivo.pdf (11.03.2015)

[8] Kervinen, A. Towards Practical Model-Based Testing: Improvements in Modelling and Test Generation. Tampere University of Technology. Publication 769. Tampere, 2008.

[9] Behrmann, G., David, A., Larsen K.G. A Tutorial on UPPAAL 4.0. [WWW] http://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf (22.03.2015)

[10] Zuker, E. UPPAAL Verification of real-time systems. [WWW] http://webcourse.cs.technion.ac.il/236800/Winter2010-2011/ho/WCFiles/ UPPAAL_Presentation.pdf (05.04.2015)

[11] OÜ ELIKO Tehnoloogia Arenduskeskuse tänavavalgustuse juhtimissüsteem [WWW] http://www.eliko.ee/public/ELIKO_SmartELI_uld.pdf (11.04.2015)

[12] Hierons, R. M., Merayo, M. G., Núñez, M. Implementation Relations for the Distributed Test Architecture. Testing of Software and Communicating Systems. Springer, 2008.

[13] Larsen, K. G., Mikucionis, M., Nielsen, B., Skou, A. Testing Real-Time Embedded Software using UPPAAL-TRON. An Industrial Case Study. [WWW] http://people.cs.aau.dk/~marius/tron/EMSOFT2005.pdf (15.04.2015)

[14] UPPAAL TRON web page. [WWW] http://people.cs.aau.dk/~marius/tron/ (16.04.2015)

[15] David, A. Larsen, K. G. More Features in UPPAAL. [WWW] https://intranet.cs.aau.dk/fileadmin/user_upload/Education/Courses/2011/SV/Uppaal3.pdf (16.04.2015)

[16] Anier, A., Vain, J. Model based continual planning and control for assistive robots. [WWW] http://dijkstra.cs.ttu.ee/~aivo/dtron/publications/healthinf-cr.pdf (16.04.2015)

[17] [WWW] http://www.uppaal.com/index.php?sida=216&rubrik=101 (25.04.2015)

[18] Larsen, K. G., Mikucionis, M., Nielsen, B., Skou, A. Testing Real-Time Embedded Software using UPPAAL-TRON an industrial case study. [WWW] http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.970&rep=rep1&type=pdf (26.04.2015)

[19] Anier, A. DTRON tutorial. [WWW] http://dijkstra.cs.ttu.ee/~aivo/dtron/dtronTutorial.pdf (26.04.2015)

[20] Markvardt, M. Tarkvara testimist käsitlev juhendmaterjal. [WWW] http://www.riso.ee/sites/default/files/Testimise_juhis.doc (28.04.2015)

[21] Roos, M. Hajussüsteemid. [WWW] http://setcom.ee/tanno/info/is/teave/ained/it/ite_com_alu_tu_roos_hajussusteemid_2009.pdf (29.04.2015)

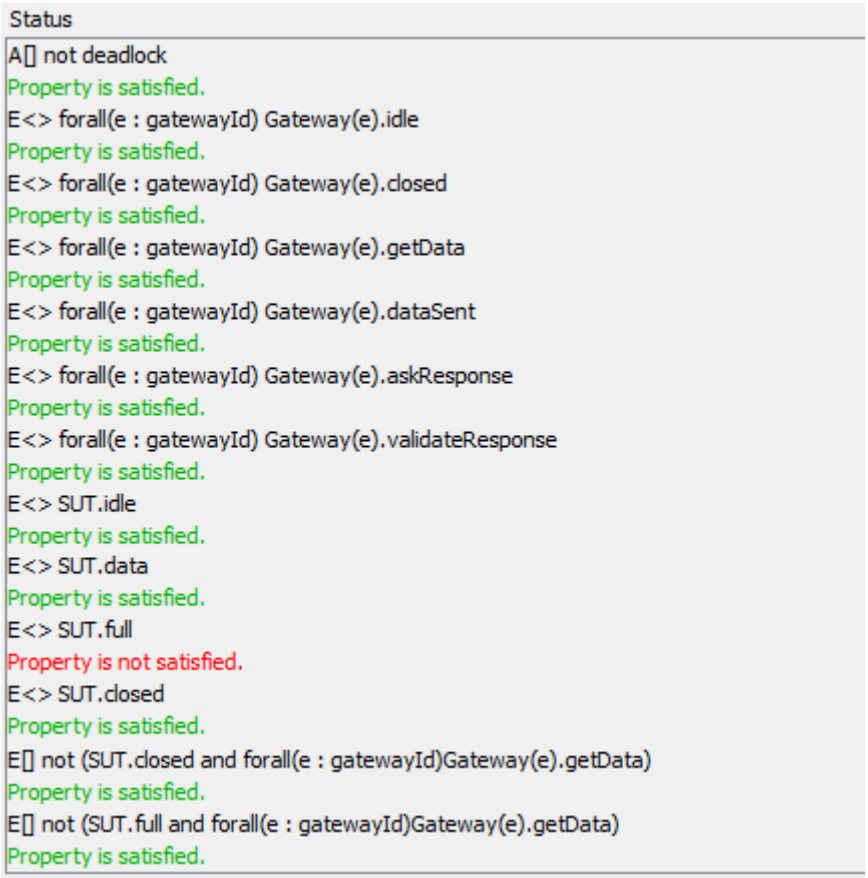
[22] DTRON web page. [WWW] http://dijkstra.cs.ttu.ee/~aivo/dtron/ (29.04.2015)

[23] E-teatmik. [WWW] http://www.vallaste.ee/ (18.05.2015)

[24] Tepandi, J. Tarkvara kvaliteet ja standardid (IDX5721, IDX5722). [WWW] http://deepthought.ttu.ee/users/tepandi/pdf/tks-loeng.pdf (18.05.2015)

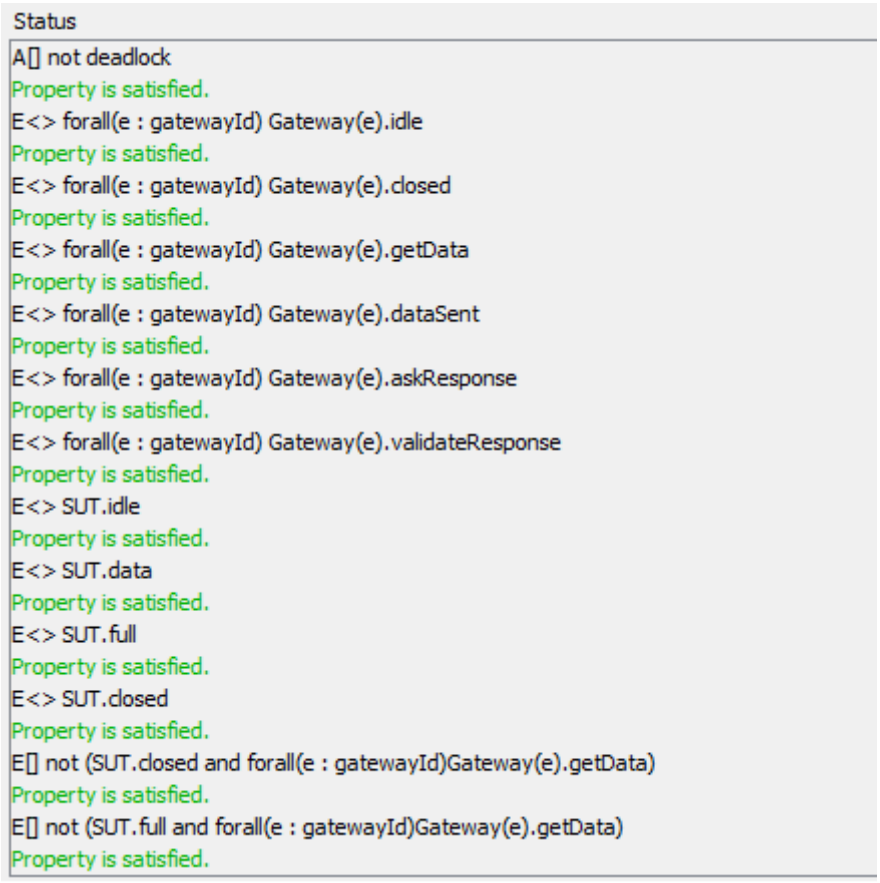
Lisa 1 – Nõuete kontrolli tulemused

Nõuete kontrolli tulemused juhul kui kilpide koguarv on väiksem kui maksimaalne lubatud ühenduste arv.



Lisa 2 – Nõuete kontrolli tulemused

Nõuete kontrolli tulemused juhul kui kilpide koguarv on suurem kui maksimaalne lubatud ühenduste arv.



Lisa 3 – Adapter

Adapter.java

```
package com.adapter;

import java.io.IOException;
import java.util.HashMap;
import java.util.Map;

import spread.*;
import ee.ttu.cs.dtron.api.domain.*;
import ee.ttu.cs.dtron.api.spread.*;

public class Adapter {

static int[] serverValuesArray = new int[5];
       int arrayValue = 0;
       final static ServerSimulator serverSimulator = new ServerSimulator();

    public static void main( String[] args ) throws SpreadException,
IOException {

       // Connect dtron
       Dtron dtron = new Dtron();
      dtron.connect();

      // Listen for channels
      IDtronChannel connectChannel = new DtronChannel("send");
      dtron.addDtronListener(new DtronListener(connectChannel) {

             @Override
             public void messageReceived(IDtronChannelValued
connectChanValue){

                    // Log incoming channel
                    System.out.println("Received: " + connectChanValue);

                    Map<String, Integer> data = new HashMap<String,
Integer>();
                    byte[] bytes = new byte[15*1024];
                    int responseValue = 0;
                    String responseVariable= "releeOlekServeris";

                    // ID-s
                    int parameetriteArv = 4;
                    int gwID = 10;
                    int releeID = 20;
                    int end = 4;
                    // Values
                    int gwNo;
                    int releeOlek;

                    // Create byte array
                    bytes[0] = (byte)gwID;

                    gwNo = connectChanValue.getVariables().get("noOfGateway");
                    bytes[1] = (byte)gwNo;
```

```
                    bytes[2] = (byte)releeID;

                   releeOlek =
connectChanValue.getVariables().get("releeOlek");
                    bytes[3] = (byte)releeOlek;

                    bytes[4] = (byte)end;

                    // Send data to server
                    for (int i = 0 ; i<=parameetriteArv; i++){
                          System.out.println("Byte array value:" +
(int)bytes[i]);
                          serverSimulator.sendData(bytes[i]);
                       }

                    serverSimulator.addState();

                    // Listen for outgoing channel
                    IDtronChannel sendDataChannel = new
DtronChannel("response");

                    serverSimulator.getGatewaySate(gwNo);
                    responseValue = getReleeValue(3);

                    data.put(responseVariable, responseValue);

                    IDtronChannelValued valued =
sendDataChannel.constructValued(data);

                    try {
                          // Log outgoing channel
                          System.out.println("Sending channel value: " +
valued);
                          getDtron().send(valued);
                          } catch (SpreadException e) {
                                e.printStackTrace();}
                    }
             });

        // Disconnect dtron (ENTER terminates SUT)
        System.in.read();
        dtron.disconnect();
    }

    public void getDataFromServer(Byte i){
      serverValuesArray[arrayValue] = (int)i;
      arrayValue++;
    }

    public static int getReleeValue(int position){
      return serverValuesArray[position];
    }

    public void setArrayLength() {
      arrayValue = 0;
    }
}
```

Lisa 4 – Testi tulemus

```
C:\Users\Age\Desktop\MBT\dtron>java -jar dtron-4.14.jar -f
streetlight.xml -u 1000 -o 400
00 -P eager
[main] INFO ee.ttu.cs.dtron.troninstaller.TronInstaller - Checking for
TRON from environment variable - TRON_HOME
[main] INFO ee.ttu.cs.dtron.troninstaller.TronInstaller - Running on
Windows, checking for "tron.exe"
[main] INFO ee.ttu.cs.dtron.troninstaller.TronInstaller - Found tron:
C:\Users\Age\Desktop\MBT\uppaal-tron-1.5-win32\tron.exe
[main] INFO ee.ttu.cs.dtron.api.spread.DtronUpta -
tronexe=C:\Users\Age\Desktop\MBT\uppaal-tron-1.5-win32\tron.exe
[main] INFO ee.ttu.cs.dtron.configuration.a - Configured with timeout
40000 and timeunit 1000
[main] INFO org.apache.commons.vfs2.impl.StandardFileSystemManager -
Using "C:\Users\Age\AppData\Local\Temp\vfs_cache" as temporary files
store.
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer -
totalNoOfGateways
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer -
totalNoOfSUTs
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer - max
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer - N
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer -
noOfConnections
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer -
i_send_noOfGateway
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer -
i_send_releeOlek
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer -
o_response_releeOlekServeris
line 3:11 token recognition error at: '[1,t'
line 3:31 token recognition error at: ']'
line 3:33 extraneous input 'gatewayId' expecting {',', ';'}
line 3:11 token recognition error at: '[0,N'
line 3:15 token recognition error at: ']'
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found channel - i_send
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found channel - o_response
[main] INFO ee.ttu.cs.dtron.configuration.d - Found incoming channel:
i_send
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
totalNoOfGateways - skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
totalNoOfSUTs - skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array: max -
skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array: N -
skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
noOfConnections - skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
o_response_releeOlekServeris - skipping
[main] INFO ee.ttu.cs.dtron.configuration.d - Found outgoing channel:
o_response
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
totalNoOfGateways - skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
totalNoOfSUTs - skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array: max -
```

```
skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array: N -
skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
noOfConnections - skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
i_send_noOfGateway - skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
i_send_releeOlek - skipping
[main] INFO ee.ttu.cs.dtron.api.spread.Dtron - Connecting to Spread at
localhost:0 (AVQv8b1K)
[main] WARN ee.ttu.cs.dtron.api.spread.Dtron - Connected to Spread at
localhost:0
[main] INFO ee.ttu.cs.dtron.api.spread.Dtron - Don't forget to clean
up and disconnect()!
[main] INFO ee.ttu.cs.dtron.api.spread.DtronUpta - Going to execute -
C:\Users\Age\Desktop\MBT\uppaal-tron-1.5-win32\tron.exe -P eager -v 9
-I SocketAdapter C:\Users\Age\Desktop\MBT\dtron\streetlight.xml --
localhost 50991
UPPAAL TRON 1.5 using UPPAAL 4.1.2 (rev. 4351), June 2009
Compiled with i586-mingw32msvc-g++ -Wall -DLIBXML_STATIC -DNDEBUG -O2
-ffloat-store -march=pentiumpro -march=pentium4 -march=prescott -
march=pentium-m -DTIGA_MERGE_STATES    -DBOOST_DISABLE_THREADS
Copyright (c) 1995 - 2009, Uppsala University and Aalborg University.
All rights reserved.
Options for UPPAAL TRON:
  Search order is breadth first
  Using no space optimisation
  State space representation uses minimal constraint systems
  Observation uncertainties: 0, 0, 0, 0 (microseconds).
  Scheduling latency: 0 microseconds
  Future precomputation: closure(0 mtu).
  Input delay extended by: 0
  OS scheduler: non-real-time.
[TR.Receiver] INFO b.a - Setting timeout to 40000, timeunit to 1000
[TR.Receiver] INFO ee.ttu.cs.dtron.api.spread.Dtron - Joining group
response
WARNING: partitioning is inconsistent, use -i option to investigate.
  Emulation invariants: Gateway(1), Gateway(2), Gateway(3),
Gateway(4), S.
  Timeunit: 1000us
  Timeout: 40000mtu
  Inputs:  i_send(i_send_releeOlek,i_send_noOfGateway)
  Outputs: o_response(o_response_releeOlekServeris)
TEST in progress - 1%[TR.Receiver] INFO
ee.ttu.cs.dtron.api.spread.Dtron - Spreading message: name=send,
variables={releeOlek=3, noOfGateway=4}
TEST in progress / 3%[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=3}
[Thread-4] INFO b.c - Reported to UPTA!
[TR.Receiver] INFO ee.ttu.cs.dtron.api.spread.Dtron - Spreading
message: name=send, variables={releeOlek=3, noOfGateway=4}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=3}
[Thread-4] INFO b.c - Reported to UPTA!
[TR.Receiver] INFO ee.ttu.cs.dtron.api.spread.Dtron - Spreading
message: name=send, variables={releeOlek=0, noOfGateway=2}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=0}
[Thread-4] INFO b.c - Reported to UPTA!
[TR.Receiver] INFO ee.ttu.cs.dtron.api.spread.Dtron - Spreading
```

```
message: name=send, variables={releeOlek=0, noOfGateway=3}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=0}
[Thread-4] INFO b.c - Reported to UPTA!
TEST in progress - 7%[TR.Receiver] INFO
ee.ttu.cs.dtron.api.spread.Dtron - Spreading message: name=send,
variables={releeOlek=0, noOfGateway=2}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=0}
[Thread-4] INFO b.c - Reported to UPTA!
[TR.Receiver] INFO ee.ttu.cs.dtron.api.spread.Dtron - Spreading
message: name=send, variables={releeOlek=3, noOfGateway=2}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=3}
[Thread-4] INFO b.c - Reported to UPTA!
[TR.Receiver] INFO ee.ttu.cs.dtron.api.spread.Dtron - Spreading
message: name=send, variables={releeOlek=1, noOfGateway=2}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=1}
[Thread-4] INFO b.c - Reported to UPTA!
TEST in progress \ 13%[TR.Receiver] INFO
ee.ttu.cs.dtron.api.spread.Dtron - Spre
ading message: name=send, variables={releeOlek=1, noOfGateway=4}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=1}
[Thread-4] INFO b.c - Reported to UPTA!
[TR.Receiver] INFO ee.ttu.cs.dtron.api.spread.Dtron - Spreading
message: name=send, variables={releeOlek=2, noOfGateway=3}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=2}
[Thread-4] INFO b.c - Reported to UPTA!
TEST in progress | 21%[TR.Receiver] INFO
ee.ttu.cs.dtron.api.spread.Dtron - Spre
ading message: name=send, variables={releeOlek=3, noOfGateway=1}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=3}
[Thread-4] INFO b.c - Reported to UPTA!
TEST in progress / 28%[TR.Receiver] INFO
ee.ttu.cs.dtron.api.spread.Dtron - Spre
ading message: name=send, variables={releeOlek=1, noOfGateway=2}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=1}
[Thread-4] INFO b.c - Reported to UPTA!
[TR.Receiver] INFO ee.ttu.cs.dtron.api.spread.Dtron - Spreading
message: name=send, variables={releeOlek=2, noOfGateway=1}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=2}
[Thread-4] INFO b.c - Reported to UPTA!
TEST in progress - 36%[TR.Receiver] INFO
ee.ttu.cs.dtron.api.spread.Dtron - Spre
ading message: name=send, variables={releeOlek=3, noOfGateway=2}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=3}
[Thread-4] INFO b.c - Reported to UPTA!
TEST in progress | 49%DRIVER: 1461093274.578832s has passed, now it's
1461093274.579831s
TEST in progress / 55%[TR.Receiver] INFO
ee.ttu.cs.dtron.api.spread.Dtron - Spre
ading message: name=send, variables={releeOlek=1, noOfGateway=1}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=1}
```

```
[Thread-4] INFO b.c - Reported to UPTA!
[TR.Receiver] INFO ee.ttu.cs.dtron.api.spread.Dtron - Spreading
message: name=send, variables={releeOlek=0, noOfGateway=3}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=0}
[Thread-4] INFO b.c - Reported to UPTA!
TEST in progress - 63%[TR.Receiver] INFO
ee.ttu.cs.dtron.api.spread.Dtron - Spre
ading message: name=send, variables={releeOlek=3, noOfGateway=1}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=3}
[Thread-4] INFO b.c - Reported to UPTA!
TEST in progress / 81%[TR.Receiver] INFO
ee.ttu.cs.dtron.api.spread.Dtron - Spre
ading message: name=send, variables={releeOlek=2, noOfGateway=1}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=2}
[Thread-4] INFO b.c - Reported to UPTA!
[TR.Receiver] INFO ee.ttu.cs.dtron.api.spread.Dtron - Spreading
message: name=send, variables={releeOlek=0, noOfGateway=4}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=0}
[Thread-4] INFO b.c - Reported to UPTA!
[TR.Receiver] INFO ee.ttu.cs.dtron.api.spread.Dtron - Spreading
message: name=send, variables={releeOlek=2, noOfGateway=2}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=2}
[Thread-4] INFO b.c - Reported to UPTA!
TEST in progress - 88%[TR.Receiver] INFO
ee.ttu.cs.dtron.api.spread.Dtron - Spre
ading message: name=send, variables={releeOlek=2, noOfGateway=3}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=2}
[Thread-4] INFO b.c - Reported to UPTA!
[TR.Receiver] INFO ee.ttu.cs.dtron.api.spread.Dtron - Spreading
message: name=send, variables={releeOlek=3, noOfGateway=2}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=3}
[Thread-4] INFO b.c - Reported to UPTA!
TEST in progress \ 95%[TR.Receiver] INFO
ee.ttu.cs.dtron.api.spread.Dtron - Spre
ading message: name=send, variables={releeOlek=2, noOfGateway=1}
[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=2}
[Thread-4] INFO b.c - Reported to UPTA!
DRIVER: 1461093292.748741s has passed, now it's 1461093292.870901s

TEST PASSED: Time out for testing
TR.Receiver: java.io.EOFException
CONN::readchannelreporter self-shutdown! (tester timed out and
disconnected?): A blocking operation was interrupted by a call to
WSACancelBlockingCall.
```

Lisa 5 – Testi tulemus ebaõnnestunud testiga

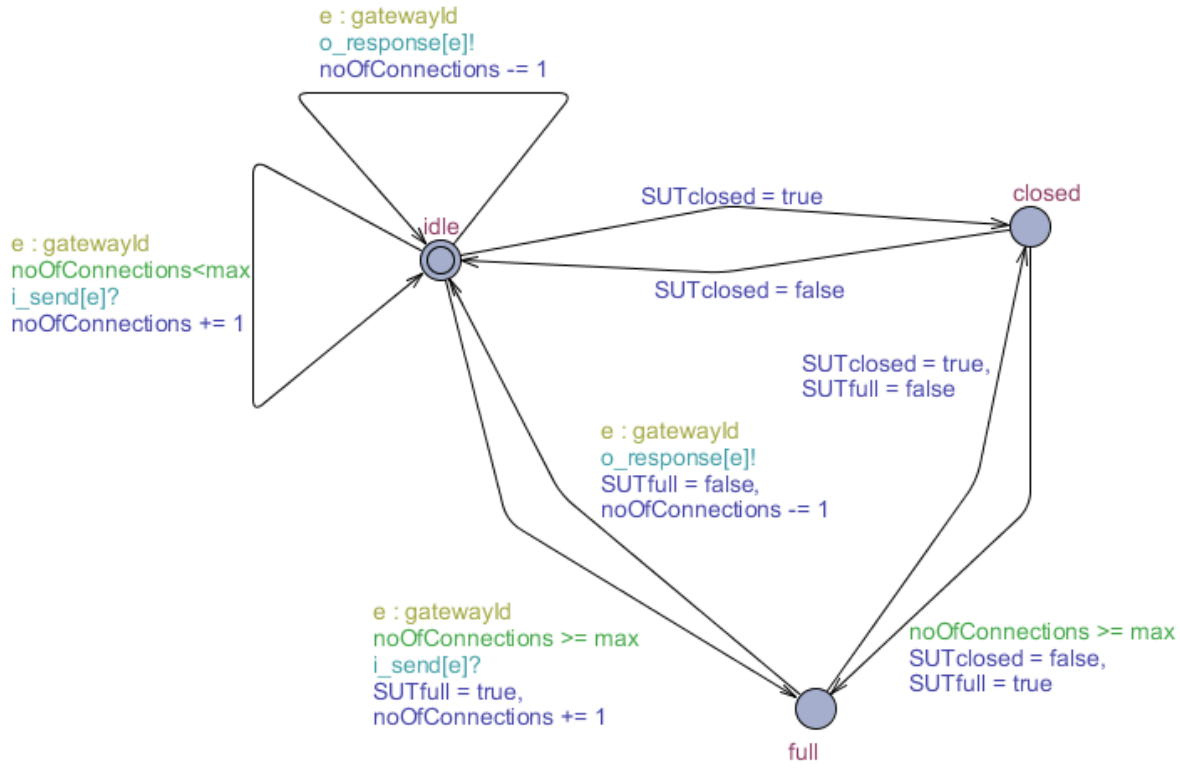
```
C:\Users\Age\Desktop\MBT\dtron>java -jar dtron-4.14.jar -f
streetlight.xml -u 1000 -o 4000 -P eager
[main] INFO ee.ttu.cs.dtron.troninstaller.TronInstaller - Checking for
TRON from environment variable - TRON_HOME
[main] INFO ee.ttu.cs.dtron.troninstaller.TronInstaller - Running on
Windows, checking for "tron.exe"
[main] INFO ee.ttu.cs.dtron.troninstaller.TronInstaller - Found tron:
C:\Users\Age\Desktop\MBT\uppaal-tron-1.5-win32\tron.exe
[main] INFO ee.ttu.cs.dtron.api.spread.DtronUpta -
tronexe=C:\Users\Age\Desktop\MBT\uppaal-tron-1.5-win32\tron.exe
[main] INFO ee.ttu.cs.dtron.configuration.a - Configured with timeout
4000 and timeunit 1000
[main] INFO org.apache.commons.vfs2.impl.StandardFileSystemManager -
Using "C:\Users\Age\AppData\Local\Temp\vfs_cache" as temporary files
store.
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer -
totalNoOfGateways
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer -
totalNoOfSUTs
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer - max
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer - N
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer -
noOfConnections
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer -
i_send_noOfGateway
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer -
i_send_releeOlek
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found integer -
o_response_releeOlekServeris
line 3:11 token recognition error at: '[1,t'
line 3:31 token recognition error at: ']'
line 3:33 extraneous input 'gatewayId' expecting {',', ';'}
line 3:11 token recognition error at: '[0,N'
line 3:15 token recognition error at: ']'
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found channel - i_send
[main] INFO ee.ttu.cs.antlrxta.AntlrXta - Found channel - o_response
[main] INFO ee.ttu.cs.dtron.configuration.d - Found incoming channel:
i_send
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
totalNoOfGateways - skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
totalNoOfSUTs - skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array: max -
skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array: N -
skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
noOfConnections - skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
o_response_releeOlekServeris - skipping
[main] INFO ee.ttu.cs.dtron.configuration.d - Found outgoing channel:
o_response
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
totalNoOfGateways - skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
totalNoOfSUTs - skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array: max -
skipping
```

```
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array: N -
skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
noOfConnections - skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
i_send_noOfGateway - skipping
[main] WARN ee.ttu.cs.dtron.configuration.d - Found int array:
i_send_releeOlek- skipping
[main] INFO ee.ttu.cs.dtron.api.spread.Dtron - Connecting to Spread at
localhost:0 (AVQwA+8T)
[main] WARN ee.ttu.cs.dtron.api.spread.Dtron - Connected to Spread at
localhost:0
[main] INFO ee.ttu.cs.dtron.api.spread.Dtron - Don't forget to clean
up and disconnect()!
[main] INFO ee.ttu.cs.dtron.api.spread.DtronUpta - Going to execute -
C:\Users\Age\Desktop\MBT\uppaal-tron-1.5-win32\tron.exe -P eager -v 9
-I SocketAdapter C:\Users\Age\Desktop\MBT\dtron\streetlight.xml --
localhost 51049
UPPAAL TRON 1.5 using UPPAAL 4.1.2 (rev. 4351), June 2009
Compiled with i586-mingw32msvc-g++ -Wall -DLIBXML_STATIC -DNDEBUG -O2
-ffloat-store -march=pentiumpro -march=pentium4 -march=prescott -
march=pentium-m -DTIGA_MERGE_STATES    -DBOOST_DISABLE_THREADS
Copyright (c) 1995 - 2009, Uppsala University and Aalborg University.
All rights reserved.
Options for UPPAAL TRON:
  Search order is breadth first
  Using no space optimisation
  State space representation uses minimal constraint systems
  Observation uncertainties: 0, 0, 0, 0 (microseconds).
  Scheduling latency: 0 microseconds
  Future precomputation: closure(0 mtu).
  Input delay extended by: 0
  OS scheduler: non-real-time.
[TR.Receiver] INFO b.a - Setting timeout to 4000, timeunit to 1000
[TR.Receiver] INFO ee.ttu.cs.dtron.api.spread.Dtron - Joining group
response
WARNING: partitioning is inconsistent, use -i option to investigate.
  Emulation invariants: Gateway(1), Gateway(2), S.
  Timeunit: 1000us
  Timeout: 4000mtu
  Inputs:  i_send(i_send_releeOlek,i_send_noOfGateway)
  Outputs: o_response(o_response_releeOlekServeris)
TEST in progress \ 8%[TR.Receiver] INFO
ee.ttu.cs.dtron.api.spread.Dtron - Spreading message: name=send,
variables={releeOlek=3, noOfGateway=1}
TEST in progress - 18%[Thread-4] INFO b.c - Received - name=response,
variables={releeOlekServeris=100}
[Thread-4] INFO b.c - Reported to UPTA!
Short post-mortem analysis based on last good stateSet(8):
1)( Gateway(1).dataSent Gateway(2).idle S.data )
S.waitResponse>720, #t>800, S.waitResponse<721, S.waitResponse-#t<=-
80, #t<801,#t-S.waitResponse<=80 noOfConnections=1
i_send_noOfGateway=1 i_send_releeOlek=3 o_response_releeOlekServeris=3
SUTclosed=0 SUTfull=0 Gateway(1).GWclosed=0 Gateway(1).gatewayId=0
Gateway(2).GWclosed=0 Gateway(2).gatewayId=0
2)( Gateway(1).dataSent Gateway(2).closed S.data )
S.waitResponse>720, #t>800, S.waitResponse<721, S.waitResponse-#t<=-
80, #t<801,#t-S.waitResponse<=80 noOfConnections=1
i_send_noOfGateway=1 i_send_releeOlek=3 o_response_releeOlekServeris=3
SUTclosed=0 SUTfull=0 Gateway(1).GWclosed=0 Gateway(1).gatewayId=0
Gateway(2).GWclosed=1 Gateway(2).gatewayId=0
```

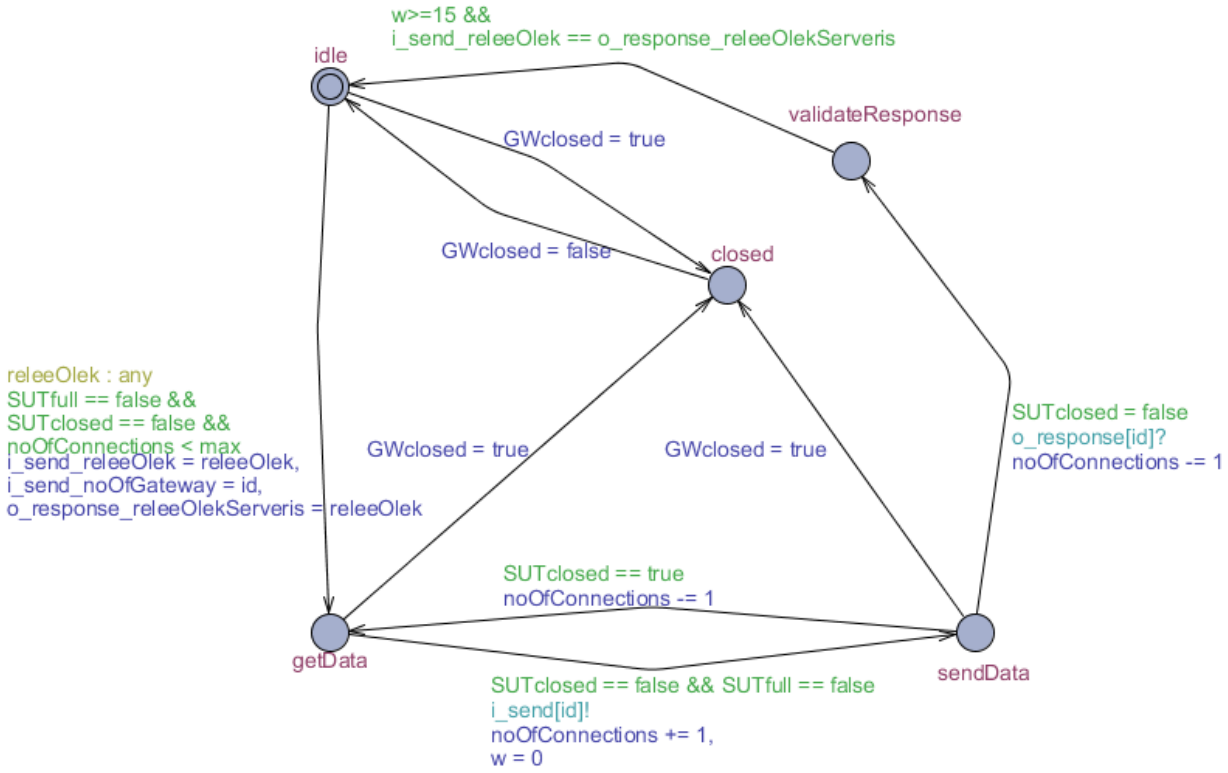
```
3)( Gateway(1).dataSent Gateway(2).getData S.data )
S.waitResponse>720, #t>800, S.waitResponse<721, S.waitResponse-#t<=-
80, #t<801,#t-S.waitResponse<=80 noOfConnections=2
i_send_noOfGateway=1 i_send_releeOlek=3 o_response_releeOlekServeris=3
SUTclosed=0 SUTfull=0 Gateway(1).GWclosed=0
Gateway(1).gateTR.Receiver: java.io.EOFExceptionwayId=0
Gateway(2).GWclosed=0 Gateway(2).gatewayId=0
4)( Gateway(1).getData Gateway(2).dataSent S.data )
S.waitResponse>720, #t>800, S.waitResponse<721, S.waitResponse-#t<=-
80, #t<801,#t-S.waitResponse<=80 noOfConnections=2
i_send_noOfGateway=1 i_send_releeOlek=3 o_response_releeOlekServeris=3
SUTclosed=0 SUTfull=0 Gateway(1).GWclosed=0 Gateway(1).gatewayId=0
Gateway(2).GWclosed=0 Gateway(2).gatewayId=0
5)( Gateway(1).dataSent Gateway(2).getData S.data )
S.waitResponse>720, #t>800, S.waitResponse<721, S.waitResponse-#t<=-
80, #t<801,#t-S.waitResponse<=80 noOfConnections=2
i_send_noOfGateway=2 i_send_releeOlek=0 o_response_releeOlekServeris=0
SUTclosed=0 SUTfull=0 Gateway(1).GWclosed=0 Gateway(1).gatewayId=0
Gateway(2).GWclosed=0 Gateway(2).gatewayId=0
6)( Gateway(1).dataSent Gateway(2).getData S.data )
S.waitResponse>720, #t>800, S.waitResponse<721, S.waitResponse-#t<=-
80, #t<801,#t-S.waitResponse<=80 noOfConnections=2
i_send_noOfGateway=2 i_send_releeOlek=1 o_response_releeOlekServeris=1
SUTclosed=0 SUTfull=0 Gateway(1).GWclosed=0 Gateway(1).gatewayId=0
Gateway(2).GWclosed=0 Gateway(2).gatewayId=0
7)( Gateway(1).dataSent Gateway(2).getData S.data )
S.waitResponse>720, #t>800, S.waitResponse<721, S.waitResponse-#t<=-
80, #t<801,#t-S.waitResponse<=80 noOfConnections=2
i_send_noOfGateway=2 i_send_releeOlek=2 o_response_releeOlekServeris=2
SUTclosed=0 SUTfull=0 Gateway(1).GWclosed=0 Gateway(1).gatewayId=0
Gateway(2).GWclosed=0 Gateway(2).gatewayId=0
8)( Gateway(1).dataSent Gateway(2).getData S.data )
S.waitResponse>720, #t>800, S.waitResponse<721, S.waitResponse-#t<=-
80, #t<801,#t-S.waitResponse<=80 noOfConnections=2
i_send_noOfGateway=2 i_send_releeOlek=3 o_response_releeOlekServeris=3
SUTclosed=0 SUTfull=0 Gateway(1).GWclosed=0 Gateway(1).gatewayId=0
Gateway(2).GWclosed=0 Gateway(2).gatewayId=0

  Options for input   : (empty)
  Options for output  :
o_response(o_response_releeOlekServeris=3)@(800;+inf),
o_response(o_response_releeOlekServeris=0)@(800;+inf),
o_response(o_respon
CONN::readchannelreporter self-shutdown! (tester timed out and
disconnected?): A blocking operation was interrupted by a call to
WSACancelBlockingCall.
se_releeOlekServeris=1)@(800;+inf),
o_response(o_response_releeOlekServeris=2)@(800;+inf)
  Options for internal: -@(800;+inf)
  Options for delay   : until +inf)
  Last time-window    : (800;801)
Got unacceptable output:
o_response(o_response_releeOlekServeris=100)@800001us a
t (800;801)
Expected outputs were:
o_response(o_response_releeOlekServeris=3)@(800;+inf),
o_response(o_response_releeOlekServeris=0)@(800;+inf),
o_response(o_response_releeOlekServeris=1)@(800;+inf),
o_response(o_response_releeOlekServeris=2)@(800;+inf)

TEST FAILED: Observed output has wrong variable value(s).
```

Lisa 6 – Parametriseeritud sünkroniseerimiskanalitega mudelid

SUT-i mudel:

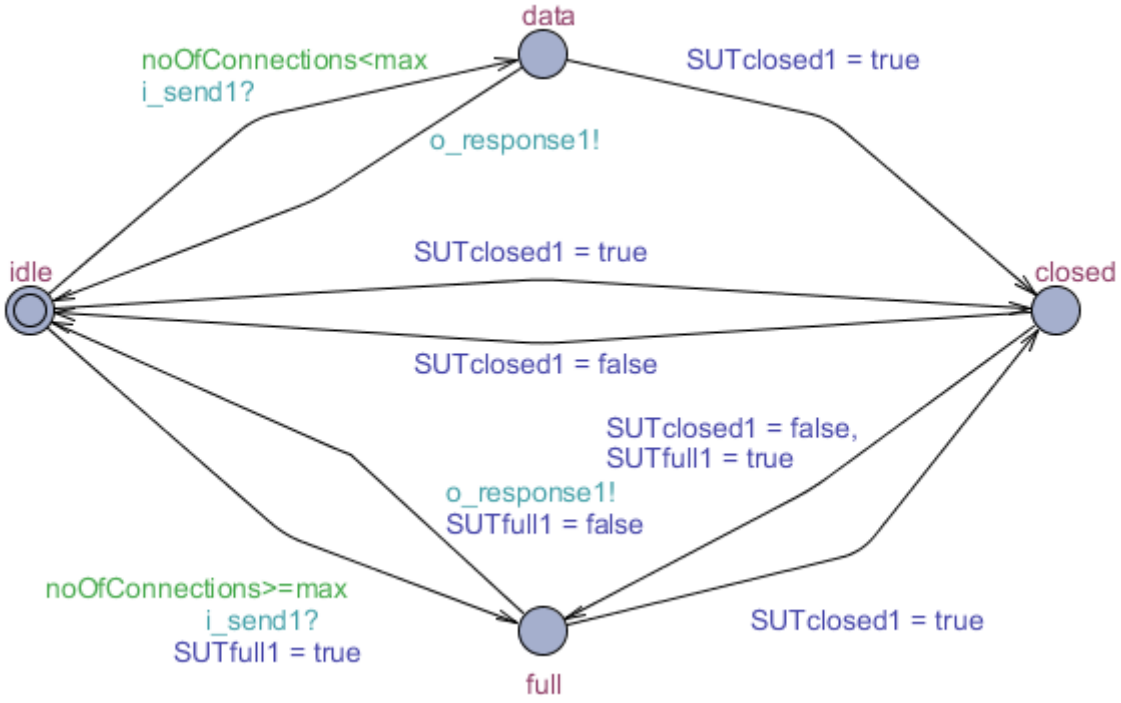


Kilbi mudel:



Lisa 7 – Individuaalseid protsesside vahelisi kanaleid kasutavad mudelid

SUT-i mudel:



Kilbi mudel: