TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond

Jürgo Pukk 155431IAPB
Siim Sams 155611IAPB
Jonas Tõnisson 154883IAPB

AJAGA AUTOMAATIDE VEEBIPÕHINE MODELLEERIMISKESKKOND

Bakalaureusetöö

Juhendaja: Gert Kanter
PhD

Tallinn 2021

Autorideklaratsioon

Kinnitame, et oleme 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.

Autorid: Jürgo Pukk, Siim Sams, Jonas Tõnisson

25.05.2021

Annotatsioon

Käesoleva lõputöö eesmärgiks oli eelmisel aastal valminud ajaga automaatide põhiste testide modelleerimise veebirakenduse edasiarendus. Veebirakendus loodi Johann Veispaki, Priit Roosivälja ja Henry Viirmäe bakalaureusetöö tiimiprojekti raames.

Veebirakenduse eesmärk on täiustada ja laiendada tellijal hetkel kasutusel olevat tarkvaralisi lahendusi UPPAAL ja ECDAR.

Töö tulemusena arendati välja projektile pidevkooste ja pidevvalmiduse keskkond, komponentide vaheliste sünkroniseerimistingimuste ja sisend-väljund operatsioonide vaade, täiendati olemasolevat graafi redigeerimise vaadet graafi elementide parema filtreerimise funktsionaalsusega ning lisati juurde võimalus avada ja muuta kuni nelja automaadi graafi samaaegselt.

Lisaks lisati tugi välisele TDL (*Test purpose specification language*) interpretaatori teenusele, millele on võimalik saata TDL keele avaldis. Avaldise põhjal genereerib interpretaator testiva automaadi mudeli.

Töös käsitletud kood on kättesaadaval GitLabi salves: https://gitlab.cs.ttu.ee/fmg/testing_tool_gui

Projekti live versioon on kättesaadaval aadressil: http://193.40.154.165/welcome

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 24 leheküljel, 8 peatükki, 8 joonist.

Abstract

Web-based Modeling Environment for Timed Automata

The aim of this thesis was to further develop a web-based modeling environment for timed automata. The web application was created last year as the result of a thesis written by Johann Veispak, Priit Roosiväli and Henry Viirmäe.

The purpose of the web application is to replace the software solutions UPPAAL and ECDAR currently in use by the client.

The result of this thesis is a continuous integration and continuous deployment environment for the project, a user interface for the synchronisation conditions between the components and the input-output operations, an upgrade of the existing graph editing view with better filtering functionality for the elements of the graph, and the possibility of concurrent opening and modifying of up to four graphs.

In addition, access was obtained to the TDL service developed internally by the university, which compiles a defined automata model and creates system models which can be tested.

All of the source code is available in GitLab repository: https://gitlab.cs.ttu.ee/fmg/testing_tool_gui

The live version of the project is available at http://193.40.154.165/welcome

The thesis is in Estonian and contains 24 pages of text, 8 chapters, 8 figures.

Lühendite ja mõistete sõnastik

API	*Application Programming Interface*, rakenduse liides teiste rakendustega liidestamiseks
CI/CD	Pidevkooste / pidevvalmidus, tarkvaraarenduse tavad muudatuste pidevaks salve laadimiseks ning kiireks automatiseeritud paigaldamiseks
CSS	*Cascading Style Sheets*, veebilehtede kujundust kirjeldavate dokumentide keel
HTML	*HyperText Markup Language*, veebilehtede sisu kirjeldavate dokumentide keel
HTTP	*Hypertext Transfer Protocol*, hüperteksti edastusprotokoll
I/O	Komponentide vahelised sisendite ja väljundite ühendused
JSON	*JavaScript Object Notation*, andmevahetusvorming
Linter	Staatilise koodi analüüsimise tööriist, mis märgistab koodi murekohad
NPM	*Node Package Manager*, JavaScripti paketihaldur
REST	*Representational State Transfer*, arhitektuuriline stiil, mis pakub standardeid võrgus olevate süsteemide vaheliseks suhtluseks [1]
SUT	*System Under Test*, testitav süsteem
Trapset	*Trapset*, kaartele määratud lõksude kogum
TDL	*Test purpose specification language*, testi eesmärgi kirjeldamise keel
XML	*Extensible Markup Language*, märgistuskeel ja faili standard
YAML	*YAML Ain't Markup Language*, inimloetav andmete serialiseerimise keel

Sisukord

Jooniste loetelu

1 Sissejuhatus

Testimine on tarkvaraarenduse protsess, kus üritatakse hinnata tarkvara nõuetele vastavust ja avastada võimalikke vigu [20]. Tavaliste süsteemide testimise puhul kontrollitakse lihtsalt sisendile vastavat väljundit. Reaalajasüsteemide puhul on sisendis lisaks veel ajalised piirangud, mida testimise käigus kontrollitakse.

Reaalajasüsteemide mudelipõhises testimises [21] luuakse süsteemist mudelid, mille eesmärk on kirjeldada süsteemi käitumist ajas. Praegu on tellijal kasutusel mudelite loomiseks UPPAAL tarkvara. UPPAAL on integreeritud tööriista keskkond reaalaja süsteemide modelleerimiseks, valideerimiseks ja kontrollimiseks kasutades laiendatud ajaga automaate ehk nn UPPAAL-i automaate.

UPPAAL on võimekas rakendus, kuid sellel on mõned puudused. Esiteks ei ole võimalik UPPAAL-is teha grupiprojekte, mis muudab ühe projekti peal kollektiivse töötamise tülikaks. Teiseks ei ole võimalik visualiseerida komponentide vahelisi sünkroniseerimistingimusi ja sisend-väljund operatsioone vaates, kus esinevad korraga mõlemad sünkroniseeritavad automaadid. Hetkel kasutatakse I/O (Sisend/väljund) visualiseerimise jaoks ECDAR rakendust. Kolmandaks ei ole võimalik UPPAAL rakenduses töötada ekraanil mitme kõrvuti visualiseeritud töölaua peal.

Eelmiste tudengite lõputöö [3] raames on arendatud veebirakendus, mis võimaldab kasutajal mudelipõhiseks testimiseks kasutatavaid mudeleid luua ning muuta. Lisaks on võimalik UPPAAL tarkvaras loodud mudeleid üle tuua käesolevas projektis loodud veebirakendusse ja vastupidi. Eelmise projekti eesmärk oli luua veebirakendus, mis pakub sama funktsionaalsust mis UPPAAL ning lisada juurde uut funktsionaalsust, mida UPPAAL ei võimalda.

Kogu kavandatud funktsionaalsust eelmine meeskond valmis ei jõudnud ning käesoleva lõputöö eesmärk on täiendada nende poolt loodud rakendust ning lahendada seal esinevad puudused.

Bakalaureusetöö meeskonna projekti kirjalikus osas on käsitletud käesoleva projekti meeskonna poolt lisatud funktsionaalsust, kasutusel olevaid tehnoloogiaid, samuti on tehtud muudatusi projekti arenduse töövoos ja eemaldatud tehnilised vajakajäämised. Lõputöö lisades on esitatud loodud tarkvara kasutusjuhend vt Lisa 2.

2 Ülesandepüstitus

Töö eesmärgiks on edasi arendada eelmisel aastal valminud lõputööd, mille raames alustati ajaga automaatide põhiste testide modelleerimiseks graafilise kasutajaliidese loomist. Kliendiga leppisime kokku projekti skoobiks järgmised punktid:

- pidevkooste ja pidevvalmiduskeskkonna (CI/CD - *Continuous integration and continuous deployment*) loomine
- komponentide vahelise sisendite ja väljundite ja sünkroniseerimis-kanalite visualiseerimise lisamine (I/O-vaade)
- välise TDL-põhise testigeneraatori teenuse integreerimine graafilisse kasutajaliidesesse
- vaadete elementide filtrite loomine ja grupeerimine
- mitme töölaua samaaegse avamise funktsionaalsus

Ülesande lahendamiseks kasutasime iteratiivset arenduse metoodikat ühenädalaste sprintidega.

3 Projekti kirjeldus

Projekt keskendub eelnevalt valminud graafilise kasutajaliidese ning tagarakenduse edasiarendusele ja tehnilise võla vähendamisele. Valminud rakendus võimaldab lisaks olemasolevale funktsionaalsusele mudeleid samaaegselt redigeerida kuni neljal eraldi töölaual, filtreerida vaadeldavaid elemente tõhusamalt, visualiseerida sisendite ja väljundite linke I/O-vaates ning genereerida TDL teenusega testi mudeleid.

3.1 Ülevaade probleemist

Eelneva projekti raames on tehtud ära suur töö rakenduse põhifunktsionaalsuse loomisel. Kasutajal on võimalik UPPAAL-is kirjutatud projekte importida loodud veebirakendusse, eksportida tagasi UPPAAL rakendusele mõistetavale kujule ning projekte muuta samaaegselt mitmel kasutajal.

Projektis käsitletud veebirakendus ei ole veel kliendil kasutusel, sest selle funktsionaalsus ei ole veel piisavalt lai.

3.2 Projekti vajalikkus

Projekt on kliendile vajalik, et muuta ajaga automaatide modelleerimise töövoog kiiremaks ja mugavamaks. Kliendi jaoks on oluline ka kontroll lähtekoodi üle, et oleks olemas võimalus lisada tulevikus juurde uut funktsionaalsust.

Mugavus saavutatakse sellega, et klient saab teha modelleerimisel paremini oma kolleegidega koostööd. Arendatud veebirakendus võimaldab mitmel kasutajal samaaegselt redigeerida projekti veebikeskkonnas. Hetkel kui klient soovib teha mingi projekti kallal koostööd peab üks inimene tegema muudatusi, eksportima projekti XML (*Extensible Markup Language*) failina ning saatma selle oma kolleegile, kes omakorda impordib XML-faili ja teeb omapoolseid muudatusi.

Lisaks on grupitöö projekti raames veebirakendus integreeritud testide eesmärgi kirjeldamise keele teenusega (TDL-teenus). Hetkel on UPPAAL rakenduses võimalik

kasutada testimise mudelite loomise eesmärgil verifitseerimise tööriista VERIFYTA [16], kuid selle kasutamine toimub läbi terminalipõhise kasutajaliidese, mis ei ole kasutajale intuitiivne ega mugav.

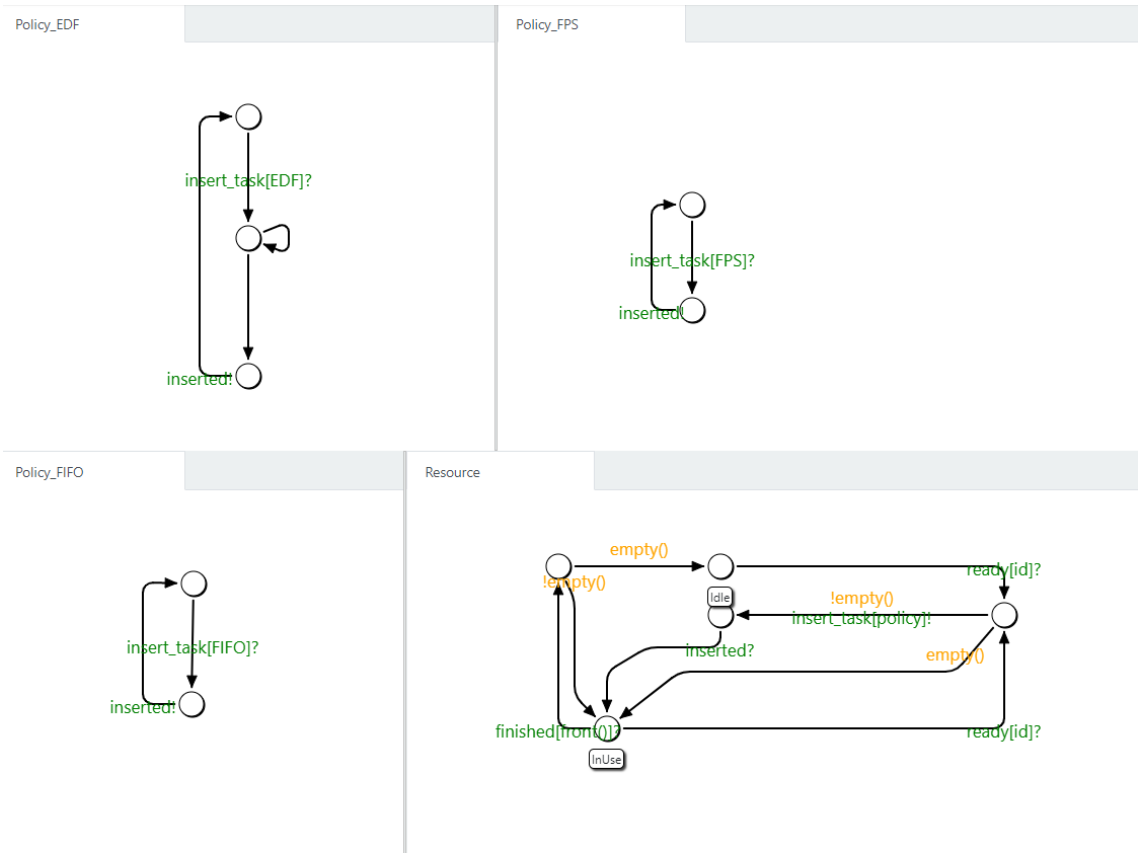
Kasutaja jaoks on oluline uuendus I/O-vaade, mis kujutab endast komponentidevaheliste sisendite ja väljundite visualiseerimist. Hetkel kasutab klient I/O visualiseerimiseks ECDAR rakendust, so tööriista, mis rakendab DLLNW10 [7] ajastatud liidese teooriat, mille arhitektuur omakorda taaskasutab UPPAAL-i põhikomponente [6].

3.3 Mudeli elementide filtreerimine

Uppaali ajaga automaatide graafiline esitus koosneb sõlmedest (UPPAAL-is *location*), suunatud kaartest (UPPAAL-is *transition*) ning kommentaaridest. Lisaks on võimalik sõlmedele määrata nimesid ning kaartele lisada *select*, *guard*, *sync* ja *update* atribuute. Vaikimisi kuvatakse kõik need elemendid kasutajaliidesesse ning neid oli võimalik erinevate vaadete loomiseks ükshaaval filtreerida. Lisatud funktsionaalsus võimaldab kasutajal projekti raames deklareerida erinevad vaated, kus saab märkida mis elemente mudeli aspektide paremaks väljatoomiseks soovitakse välja filtreerida ja mida visualiseerida.

3.4 Graafi redigeerimise töölaud

UPPAAL-is on võimalik avada üks mudel korraga. Eelmise projekti käigus loodi võimekus kahe mudeli samaaegselt kuvamiseks ning redigeerimiseks. See saavutati vastava valiku korral töölaua horisontaalselt poolitamisega. Käesoleva projekti käigus sai laiendatud seda funktsionaalsust kuni nelja korraga avatavale mudeli automaadile, vt Joonis 1. See saavutati lisanduva vertikaalse poolitamisega, mida rakendatakse, kui kaks mudelit on juba avatud. Samuti sai loodud võimekus mudeleid erinevate poolituste vahel liigutada. See toimib vaates mudeli kaardil vastavale nupule vajutades ning seejärel mõnele teises vaates ilmuvale nupule vajutades.

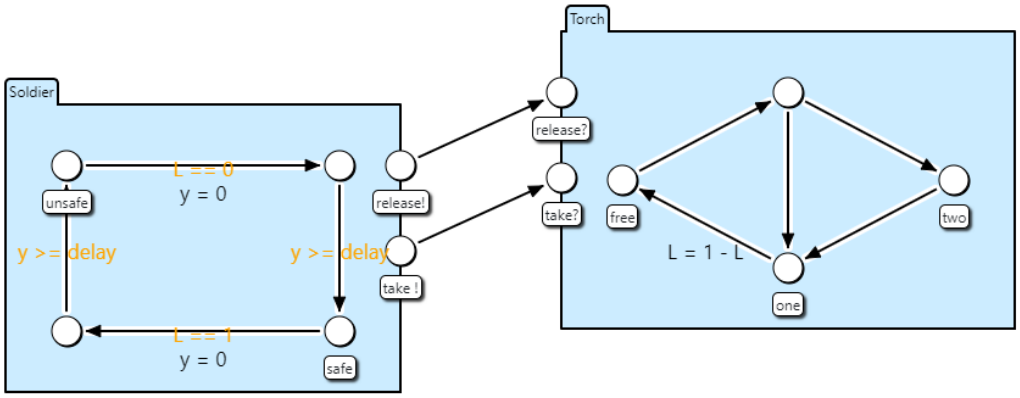


Joonis 1.Graafi redigeerimise töölaud nelja mudeliga

3.5 I/O-vaade

Visualiseerimaks mudeli automaatide sisendite ja väljundite üksteisele vastavust loodi I/O-vaade. Selles vaates joonistatakse testitavad mudelid erinevatesse akendesse, mis kannavad avatava mudeli nime. Mudeli aknasse kuvatakse ka kõik sõlmed, suunatud kaared ning atribuudid, millest see koosneb. Erandiks on *sync* atribuut, mida kasutatakse

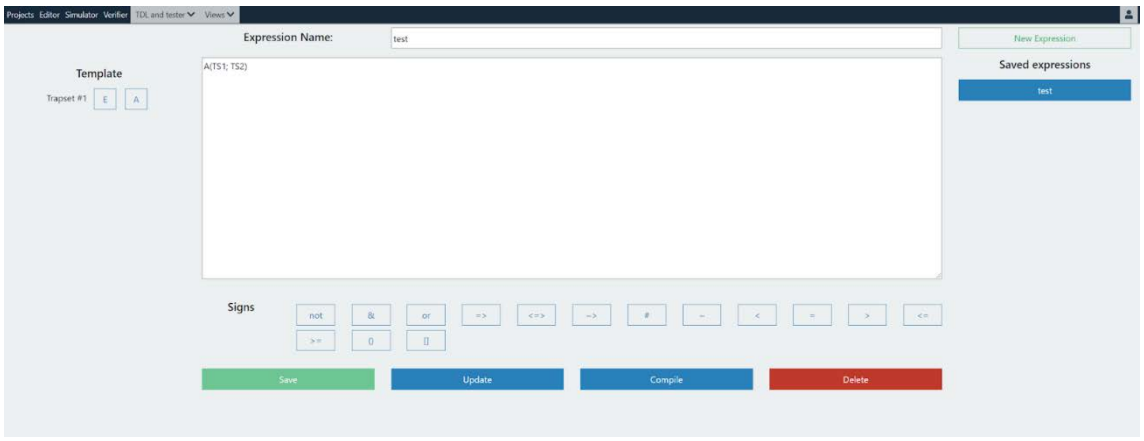
automaadivaheliste seoste loomisel. Selle atribuudi väärtust on võimalik välja lugeda kanalitelt, mis ühendavad automaatide kaari, ning märgistavad andmeid, mis mudelite vahel liiguvad. Iga mudeli kanaliga seotud kaare külge genereeritakse eelneva defineerimise korral vastavate kanalite sisendid või väljundid, vt Joonis 2. Kanalid on ühesuunalised. See tähendab, et väljundid on kanalite läbi ühendatud teiste mudelite sisenditesse. Lisaks on vaates võimalik defineerida partitsioone, mis aitavad visuaalselt grupeerida erinevaid projekti mudeleid. Nii saab näiteks märgistada testitavat süsteemi ehk SUT (*System Under Test*) ümbritsedes selleks kõik testitavad mudelid grupi nime kandva partitsiooniga.



Joonis 2. I/O-vaate ühendused

3.6 TDL-redaktori vaade

TDL-redaktori vaade on testi eesmärgi kirjeldamiseks mõeldud vaade, vt Joonis 3. Testi eesmärk koosneb loogikaelementidest ja lõksudest, millega on võimalik defineerida elementide läbimise kulgu. Vaates on võimalik kasutada eelnevalt defineeritud lõksude kogumeid, valida loogika elemente, tehtut salvestada ja muuta. Selles töös lisati võimekus peale eesmärgi defineerimist ja salvestamist välise teenuse abil testid kompileerida, mis salvestatakse automaatselt olemasoleva mudeli asemele.



Joonis 3. TDL-redaktori vaade

3.7 XML-parser

XML-parser on rakenduses olulise tähtsusega, sest UPAAL-i poolt kasutatav faili tüüp on XML. Kõik meie rakenduses olev kasutab JSON (*JavaScript Object Notation*) formaati, mille manipuleerimine on Javascripti sisse ehitatud. Selleks, et me saaksime teisendada mudeleid XML formaadist JSON formaati ja tagasi, loodi *Parser* teenus. Teenus peab olema täpne, et faili sisu ei moonduks ja ei kaoks failis sisalduvat infot.

Lisaks luuakse nüüd projekti eksportimisel UPPAAL-iga ühilduvale XML-failile ka teine JSON-formaadis fail. Teine fail sisaldab meie tööriista poolt loodud spetsiifilist infot, mida UPPAAL ei käitle. Selle vajalikkus tekkis asjaolust, et eksporditud projekti UPPAAL-i importimise ning sealt taas eksportimise käigus hüljatakse kõik sellele rakendusele võõrad sildid, kommentaarid ja omadused. Ka importimisel on kasutajal nüüd võimalik lisada teine spetsiifikat sisaldav fail, mille sisu laetakse andmebaasi ning seotakse uue loodud projektiga.

4 Projekti disain

Antud peatükis räägitakse milliseid tehnoloogilisi ja arhitektuurilisi valikuid on tehtud projektis ja põhjendatakse miks sellised valikud tehti.

4.1 Projekti arhitektuur

Varasemalt oli tarkvaraarenduses populaarne monoliitne arhitektuur. Monoliitse arhitektuuri puhul jooksevad kõik projekti teenused ühes veebiserveris ja on tihti omavahel läbipõimunud. Antud arhitektuuriga kaasneb mitu miinust. Monoliitseid rakendusi on raske horisontaalselt skaleerida ja tavaliselt minnakse vertikaalse skaleerimise teed [4]. Koodibaasi põimudes tekib olukord, kus üks vigane teenus tekitab kaskaadi ja halvab kogu rakenduse. Kuna rakendus on suur, siis võtab uue instantsi ülespanek mitu minutit aega.

Meie projekt realiseeriti kasutades mikroteenuste arhitektuuri. Mikroteenuste arhitektuuri puhul jooksevad kõik teenused eraldi veebiserverites. Neid teenuseid on võimalik individuaalselt arendada, testida ja käivitada. Mikroteenuseid kasutavat projekti on kerge horisontaalselt skaleerida [5]. Kuna teenused jooksevad eraldi veebiserverites, siis teeb see üleliigse põimumise pea võimatuks ja sunnib teenustega suhtlemiseks kasutama REST-i (*Representational State Transfer*) või sõnumivahetuse teenust. Kuna teenused on pisikesed, siis on probleemi tekkides võimalik kiiresti vigane teenus korras teenusega asendada.

4.2 Projektis kasutatavad tehnoloogiad

Suures osas on kasutatavad tehnoloogiad valitud eelneva tiimi poolt. Kuna nende tehtud valikute vastu meil pretensioone ei olnud, siis ei pidanud me vajalikuks ka neid valikuid muuta.

Eelnev meeskond valis programmeerimiskeeleks avatud lähtekoodiga TypeScripti [18]. TypeScript on Javascripti ümber olev kiht, mis kompileeritakse puhtaks Javascriptiks.

TypeScript lisab klassi ja mooduli toe, staatilise tüübi kontrolli, ES6 karakteristiku toe [19], selge teegi ja API (*Application Programming Interface*) definitsiooni võimekuse ja palju muud.

Serveripoolne kood jookseb avatud lähtekoodiga Node.js keskkonnas. Node.js võimaldab jooksutada Javascripti väljaspool brauserit.

Andmebaasina kasutame PostgreSQLi, mis on avatud lähtekoodiga laialt kasutatud relatsiooniline andmebaasisüsteem.

Dockerit kasutame mikroteenuste jooksutamiseks ühes masinas. Docker tagab teenuste isoleerituse ja garanteerib, et konteiner töötab igas toetatud keskkonnas samamoodi. Rakenduse käivitamiseks kasutame Docker Compose-i, mis võimaldab kõiki teenuste konteinereid korraga käivitada ja säilitab konteinerite andmed taaskäivitamise korral.

Suhtlus mikroteenustega ja staatiliste failide serveerimine käib läbi avatud lähtekoodiga NGINX-i. NGINX on veebi serveerimise, tagurpidi proksi, vahemälu, koormuse jaotamise, meedia striimimise ja muude võimekustega.

4.2.1 Raamistikud

4.2.1.1 Vue.js

Javascripti raamistik, mis on tänu oma modulaarsusele väikese jalajäljega ning kombineerib mitme eelneva populaarse raamistiku filosoofiat ja kirjapilti. See teeb selle väga heaks tudengiprojekti raamistiku valikuks, sest võimaldab kõigil kes on varasemalt mingisuguse javascripti raamistikuga kokku puutunud, kiiresti produktiivseks muutuda. Kasutame seda graafilise kasutajaliidese muutmiseks.

4.2.1.2 BootstrapVue

BootstrapVue on disaini raamistik Vue rakenduste jaoks, mis on ekraani suurusele reageerivate reeglite ja ilustatud komponentide kogum [17]. Tegu on Vue ümbrisega Bootstrapi ümber, mis võimaldab kiirelt Vue projektis Bootstrappi kasutada [10]. Kasutame seda graafilise kasutajaliidese stiliseerimiseks.

4.2.1.3 Jest

Jest on testimise raamistik, mis on kiire ja genereerib selgeid ja ilusaid testitulemuste raporteid [11]. Kasutame seda enda testide jooksutamiseks.

4.2.2 Teegid

4.2.2.1 P5.js

HTML (*HyperText Markup Language*) lõuendiga töötamiseks abistavate meetodite kogum [12]. Rakenduse põhifunktsionaalsus ehk mudeli graafiredaktor baseerub selle teegi meetoditel.

4.2.2.2 Vuex

Tegu on Vue mooduliga, mis lisab Vuele tsentraalse isoleeritud komponentide võimekuse, millega reeglid tagavad, et komponendi olek muutub ettearvatavalt [13].

4.2.2.3 Koa.js

Minimaalne ja paindlik Node.js HTTP (*Hypertext Transfer Protocol*) serveri teek [14]. Kasutame seda rakenduses teenuste suhtluse implementeerimiseks.

4.2.2.4 SuperTest

HTTP päringute vastuste ja staatuste kontrollimiseks mõeldud teek [15]. Kasutame seda koos Jestiga integratsioonitestide jaoks.

4.3 Alternatiivid

Graafilises rakenduses oleks saanud kasutada mitmeid erinevaid javascripti raamistikke nagu näiteks Angular, React, Ember, Backbone. Kõigil neil on omad plussid ja miinused, aga olulisemad põhjused, miks neid ei valitud on, et Vue on väga aktiivse kogukonnaga, hea dokumentatsiooniga, kiiresti õpitav ja väga paindlik.

CSS (*Cascading Style Sheets*) raamistikke on äärmiselt palju. Neist mõned alternatiivid on Foundation, Tailwind, Semantic UI ja Bulma. Mõned neist on oma failide poolest kompaktsemad kui meie projektis kasutusel olev Bootstrap, mis on pigem suurte failidega. Bootstrap, mis oli üks esimesi suuri raamistikke ja omab endiselt suurt kasutajaskonda annab aga kindlustunde, et seda uuendatakse ka tulevikus.

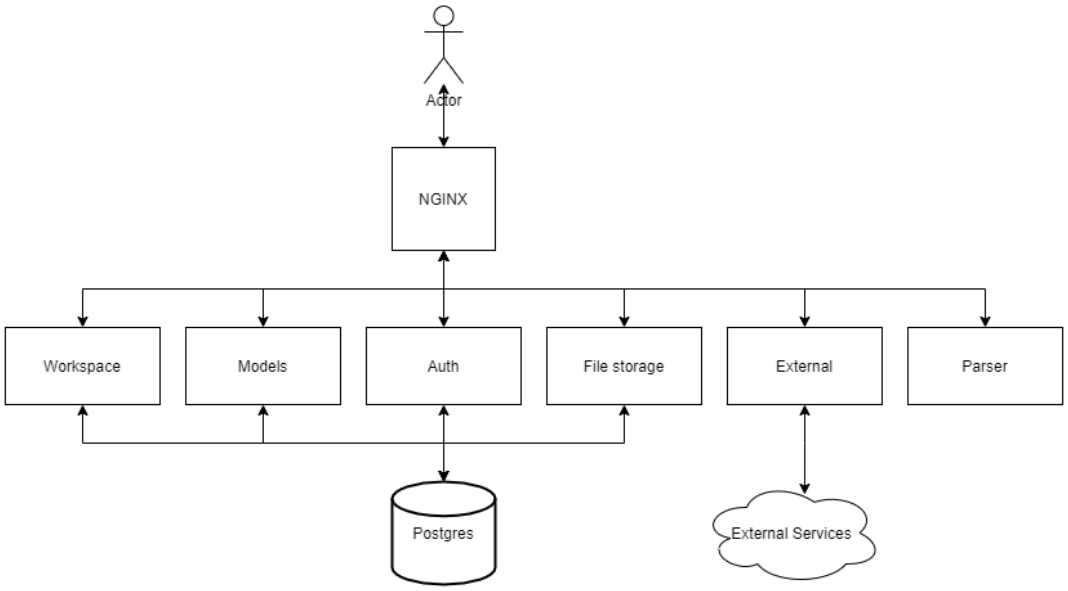
HTML lõuendi manipuleerimiseks on mitmeid alternatiive. Näiteks Draw2D, mis omab suurt sisseehitatud võimekust, mida meie ja eelneva tiimi projekti käigus ise implementeeriti. Kuna eelnev implementatsioon oli tehtud kasutades P5.js-i kasutades ja soovisime erinevates vaadetes sama stiili hoida, siis jätkasime selle kasutamist.

5 Projekti sisu

Arendatud süsteem jaguneb kaheks osaks, kasutajaliides ja tagarakendus. Kasutajaliides koosneb neljast põhivaatest ning tagarakendus koosneb kuuest mikroteenusest.

Kasutajaliideses olid varasemalt realiseeritud projektivaade, graafi redigeerimise vaade ja TDL-redaktori vaade. Grupitöö raames sai lisatud juurde I/O-vaade, täiendatud graafi redigeerimise vaadet ning integreeritud TDL-redigeerimise vaatele TDL-teenusele kompileerimise päringu esitamise funktsionaalsus.

Tagarakendus koosneb *Workspace*, *Model*, *Authorisation*, *File Storage*, *Parser* ja *External* teenustest, nagu on näha Jooniselt 4. *External* teenus on projekti raames lisatud teenus, mis suhtleb testi eesmärgi kirjeldamise keele TDL-rakenduse liidesega, mis on arendatud väljaspool meie grupitöö projekti.



Joonis 4. Tagarakenduse arhitektuuri joonis

5.1 Kasutajaliides

Projekti kasutajaliideses on viis põhivaadet. Projektide vaade, graafi redigeerimise vaade, I/O-vaade, TDL redigeerimise vaade ja testimise simuleerimise vaade. Viimase funktsionaalsuse arendamine ei mahtunud selle projekti skoopi.

Kasutajaliides on kirjutatud kasutades Vue.js raamistikku koos TypeScriptiga. TypeScripti oli kasutusele võetud eelmise meeskonna poolt, et tagada koodi tugev tüüpimine, mis aitaks vältida vigase koodi kirjutamist ning hõlbustaks probleemide tuvastamist.

Arendusega alustades ilmnes tõsiasi, et kasutajaliidese koodis kasutatakse TypeScripti ja Javascripti läbisegi. Koodistiili kontroll luges seda veaks. Seetõttu oli mingi hetk eelneva tiimi poolt tehtud otsus, et koodistiili kontrolli antud vigu ignoreeritakse. Kuna projekti skoopi kuulus ka muudatuste pidevkooste ja pidevvalmidus, siis osutus see probleemiks, mis oli vaja lahendada, kuna vastasel juhul ei olnud võimalik lisada automatiseeritud protsesse, mis kontrollivad koodi stiili ja süntaksi vigu.

Esialgu eemaldati vigade lahendamiseks kõik *linteri* (Staatilise koodi analüüsimise tööriist) ignoreerimise käsud ja jooksutati *linterit* parandamise režiimis, mis suudab tuvastada lihtsamaid vigu ning need automaatselt kõrvaldada. Pärast seda tekkis projekti kompileerimisel sadu probleeme, millega pidi käsitsi tegelema. Projekt sai edukalt üle viidud TypeScripti peale, kuid selleks kulus kümneid arendustunde ning projektis säilisid veel linteri hoiatavad sõnumid, kuna nende kõikide lahendamise peale oleks kulunud liiga palju aega.

5.1.1 Graafi redigeerimise vaade

Graafi redigeerimise vaatele on võimalik navigeerida projekti avades. Külgmenüüs on välja toodud automaatide mudelid, deklaratsioonid, lõksud, automaadi elementide kihid ning mudeli elementide filtreerimise grupeerimise jaoks loodud vaated. Igale projektile on võimalik lisada graafi mudeleid ning mudelitele on võimalik lisada sõlmi ja kaari.

Igal mudelil on vaja defineerida algussõlm, et automaati testides oleks võimalik aru saada, kust peab simulatsioon pihta hakkama. Igale sõlmele on võimalik lisada nimi ja *invariant* atribuut, mis on tingimus, mis peab olema täidetud, et liikuda edasi järgmisesse sõlme [9].

Sõlmi ühendavad kaared. Kaartel on võimalik määrata neli atribuuti: *select*, *guard*, *sync* ja *update*. *Select* on atribuut, mis sisaldab endas komadega eraldatud nimekirja tüübi avaldistest, kus "*name*" on muutuja nimi ja "*type*" on mingi defineeritud tüüp. Need muutujad on kättesaadavad ainult kaarel kus nad on defineeritud ja nad võtavad mitte-deterministliku väärtuse oma vastavate tüüpide valikus [9].

Guard on avaldis, mille tulemus on binaarne ning see määrab tingimused kaare läbimiseks [9].

Sync on atribuut, millega on võimalik defineerida komponentide vahelisi sisendite ja väljundite ja sünkroniseerimiskanaleid. *Sync* atribuudi väärtus võib olla ainult kujul "expression!", "expression?" või võib olla tühjaks jäetud [9].

Update atribuut koosneb komadega eraldatud avaldistest ning see määrab ära kaare läbimisel täidetava tegevuse [9].

5.1.1.1 Graafi elementide filtreerimine

Graafi redigeerimise vaate funktsionaalsusele on juurde lisatud kaarte ja sõlmede ja nende atribuutide filtreerimise grupeerimine. Varasemalt oli võimalik ainult ühe kaupa valida mis elemente soovib kasutaja antud vaatest välja filtreerida. Projekti käigus sai juurde arendatud funktsionaalsus, mis võimaldab lisada projektile uusi vaateid. Vaateid on võimalik lisada projektile külje menüüst "*View*" valikust. Pärast vaate loomist saab vaadet muuta. "*Edit view*" valimisel avatakse kasutajale modaal, kus on võimalik vaatele määrata äratuntav nimi, kirjeldus ja valida kõik elemendid, mida kasutaja soovib vajadusel graafi redigeerimise vaatest eemaldada.

Vaate salvestamisel talletatakse kõik valikud JSON formaadis andmebaasi ning loodud vaade on seotud projektiga. Esialgselt oli plaanis vaateid rakendada graafi mudelitele eraldi, kuid selline muudatus oleks muutunud liiga mahukaks, kuna selle funktsionaalsuse saavutamiseks oleks pidanud hakkama oluliselt muutma juba olemasolevat kasutusloogikat. Hetkel toimib elementide filtreerimine objektide kaupa. Näiteks, kui soovitakse eemaldada vaatest kõik graafi sõlmede atribuudid, siis taustal toimib see nii, et filtreerimisel muudetakse kõik sõlmede atribuut tüüpi objektid nähtamatuks.

Teiseks suuremaks takistuseks funktsionaalsuse arendamisel oli andmebaasi muudatuste tegemine. Ajutiselt loodi tabel manuaalselt läbi PgAdmin liidese, kuid kui vaate

funktsionaalsus oli valmis, pidi kirjutama migratsiooni skripti uue tabeli jaoks, et rakenduse pideval tarnimisel jõuaksid kõik muudatused ka *live* keskkonda.

Migratsiooni jooksutamisel oli peamiseks probleemiks projekti vähene dokumentatsioon ning puudusid juhised migratsiooni skriptide muutmiseks ja loomiseks. Teenuste samm-sammulise läbimise juures ei tulnud ka probleem esile, kuna veateated ei andnud edasi rohkem informatsiooni kui see et migratsiooni jooksutamine ebaõnnestus. Probleem seisnes lõpuks selles, et kõik migratsiooni skriptid räsitakse andmebaasis pärast esialgset jooksutamist ning iga kord kui skripte uuesti jooksutatakse kontrollitakse hetkelist räsi andmebaasis olevaga, et üks migratsiooni skript mitu korda ei jookseks ja andmebaasis vigasid ei tekiks. Lahenduseks oli räside kustutamine tabelist ning migratsioonid toimisid nagu oli ette nähtud.

5.1.1.2 Graafi redigeerimise töölauad

Graafi redigeerimise vaates on võimalik mudeleid avada kahel viisil. Avades mudeli mitmikvaates läbi kontekstmenüü alampunkti "*Open in split view*" poolitatakse graafi redigeerimise töölaud kaheks. Nii luuakse kaks üksteisest püstkriipsuga eraldatud BootstrapVue kaartide komponenti, mida siin nimetame poolituseks ning millest mõlemad kannavad p5.js lõuendit. Valides ühes poolituses mõne selles avatud kaardi joonistatakse kaartide valiku all lõuendile vastava automaadi mudeli graaf.

Selle projekti raames sai töölaua mitmikvaate võimekust täiendatud. Nüüd on kasutajal võimalik redigeerida kuni nelja mudelit korraga, milleks poolitatakse töölaud ka horisontaalselt. Selleks taaskasutati eelnevas projektis loodud poolitatud vaate komponenti. Kuna tegemist oli jäiga implementatsiooniga, tuli selle käitumine ümber kirjutada. Pidades arvet iga automaadi avatuse olekust ning poolitusest, milles see viimati asus, on võimalik selgeks teha mitu poolitust parajasti avatud peab olema. Seda saab ära kasutada mitme äärejuhtumi puhul, näiteks mõne vahepealse poolituse sulgemisel, kui on vaja järgnevad mudelid ühe poolituse võrra tagasi liigutada või käivitades samasuguse olukorra mõne juba avatud mudeli taasavamise korral.

Kui mudeli avamisel on võimalusteks vaid tavapärane avamine või avamine uues poolituses, on hädavajalik ka mudelite automaatide liigutamine erinevate poolituste vahel. See sai lahendatud implementatsiooniga, mille puhul tuleb vajutada hiire kaardi kohale liigutamisel sellele tekkiva nupu peale, mille järel luuakse teistes poolitustes

viimasest kaardist paremale maandumise nupp. Vajutades ühele mitmest maandumise nupust, peidetakse tekkinud nupud, liigutatakse mudeli uuele poolitusele ning juhul, kui tegemist oli poolituse viimase kaardiga, liigutatakse ka järgnevad kaardid eelmisele poolitusele.

5.1.2 I/O-vaade

Visualiseerimaks mudelite vahelist infoliikumist realiseeriti vaade, milles kujutatakse projektis defineeritud mudeleid ning nendevahelisi seoseid. Seosed on kanalite kujul, kanaleid kasutatakse omakorda väärtuste edastamiseks mudelist mudelisse.

5.1.2.1 I/O-vaate avamine

I/O-vaade on ligipääsetav, kui kasutaja on avanud soovitud projekti. Seejuures tuleb valida ülemise menüüriba pealt "*Views*" ning avanevast rippmenüüst valik "*I/O*". Selleks on kasutatud Vuex oleku haldamist. Talletades kasutaja valiku, kuvatakse redigeerimise aknas vaid antud vaatele olulisi komponente ning muudetakse nuppude käitumist vastavalt. Tööriistaribalt on I/O-vaates eemaldatud nupud sõlmede ning kaarte lisamiseks. Vasakus menüüs mudelile paremklõpsuga vajutamisel avatavast kontekst menüüst on samuti eemaldatud võimalus seda poolituses avada.

5.1.2.2 Mudelite avamine I/O-vaates

Avades mistahes projektiga seotud mudeli, joonistatakse p5.js lõuendile mudelit kujutav ristkülikukujuline konteiner. Konteineri dünaamiliseks suuruse määramiseks leiame mudelil defineeritud sõlmede seast x- ja y-telje miinimum- ja maksimumväärtused millega saame välja arvutada konteineri vajaliku suuruse.

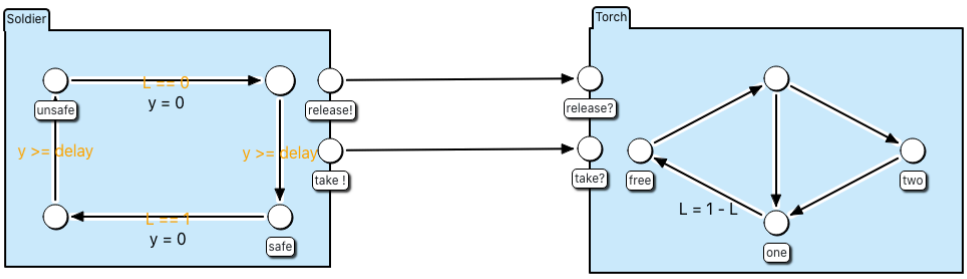
Seejärel joonistatakse lõuendile automaadi graaf selliste koordinaatidega, et see jääks konteineri piiridesse. Seejuures on kasutajal eemaldatud võimalus muuta sõlmede ning kaarte asukohta graafil. Samuti lisatakse mudeli nimi sildina konteineri ülemisse äärde. Avades seejärel teise mudeli, joonistatakse selle sarnase konstruktsioonina samale lõuendile vabale kohale esimese mudeli kõrval.

Mudeli automaadi kaartele on lisatud sisend- ja väljund elemendid, mis luuakse kaartel defineeritud *sync* atribuudi järgi. See atribuut määrab ära komponentide vahelise sisendite ja väljundite sünkroniseerimis-kanalid. Kui sync atribuut on defineeritud järelliitega "!", siis tähistab see väljundit, kui atribuut on defineeritud järelliitega "?", siis tähistab see

sisendit. Sünkronisatsiooni atribuudi põhjal genereerime suunatud kanalid mudelite vahele.

Kanalite genereerimine toimub vastavalt mudeli lisamisele I/O-vaatesse, vt Joonis 5. Esimese mudeli puhul genereeritakse kõik sisend- ja väljundkanalite väärtused mudeli konteineri paremale küljele ning teise lisatud mudeli puhul vasakule küljele. Sama loogika kehtib ka kõikide järgnevate mudelite lisamisel.

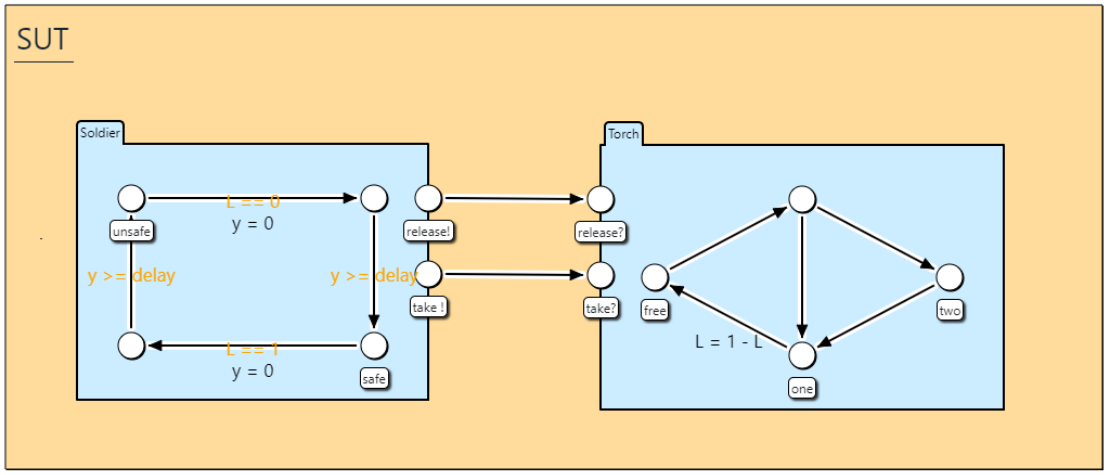
Kanalid joonistatakse esialgu sisendi ja väljundi vahele sirgjooneliselt, mis toimib kahe mudeli puhul väga hästi, kuid keerulisemate süsteemide puhul jääb see lahendus puudulikuks. Kui mudeleid,mille vahel kanalid jooksevad, on rohkem, peab kasutaja manuaalselt kanalite teekonda redigeerima. Seda on võimalik teha kanalile murdepunkti lisamisega vajutades kanali peale hiire rullikuga. Murdepunktide lisamisel ei ole mingit piirangut ning sellega on võimalik visuaalselt saavutada see eesmärk mida kasutaja soovib. Parem lahendus oleks olnud dünaamiliselt genereerida mudelite vahelised kanalid, mis väldivad kokkupõrkeid kõikide teiste elementide vahel, kuid kahjuks ei jõutud projekti raames sellise funktsionaalsuse arendamiseni.



Joonis 5. I/O-mudelite kanalid

Pärast mitme mudeli avamist on võimalik neid grupeerida visualiseerimaks erinevaid testimise komponente. Sellist gruppi märgistatakse mudelite ümber joonistatud kastiga ning nimetatakse partitsiooniks. Näide partitsioonist on SUT (*System under test*) ehk testitav süsteem, millesse võib kasutaja tahta hõlmata iga töölaual lahti oleva mudeli, mis osaleb testimises, vt Joonis 6. Piiritledes SUT-partitsiooniga vastavad mudelid on muuhulgas kergesti märgatav millised kanalid on süsteemisisesed ning millised suhtlevad

väliste komponentidega. Olles loonud töölauale partitsiooni on kasutajal võimalik neid liigutada, ümber nimetada või kustutada.



Joonis 6. Partitsioon I/O-vaates

Olles eelnevalt mõne mudeli avanud ning avades projekti I/O-vaate teist korda, taastatakse eelnev mudelite asukoht andmebaasi talletatud andmete põhjal. Lisaks taasluuakse kõik eelnevalt loodud partitsioonid. Talletatakse ka I/O-vaate mineviku, hetke ning tuleviku olekut, et saaks töölaual tegutsedes toimingud tagasi võtta või taasteostada, kuid vastav funktsionaalsus ei mahtunud tehtud töö skoopi.

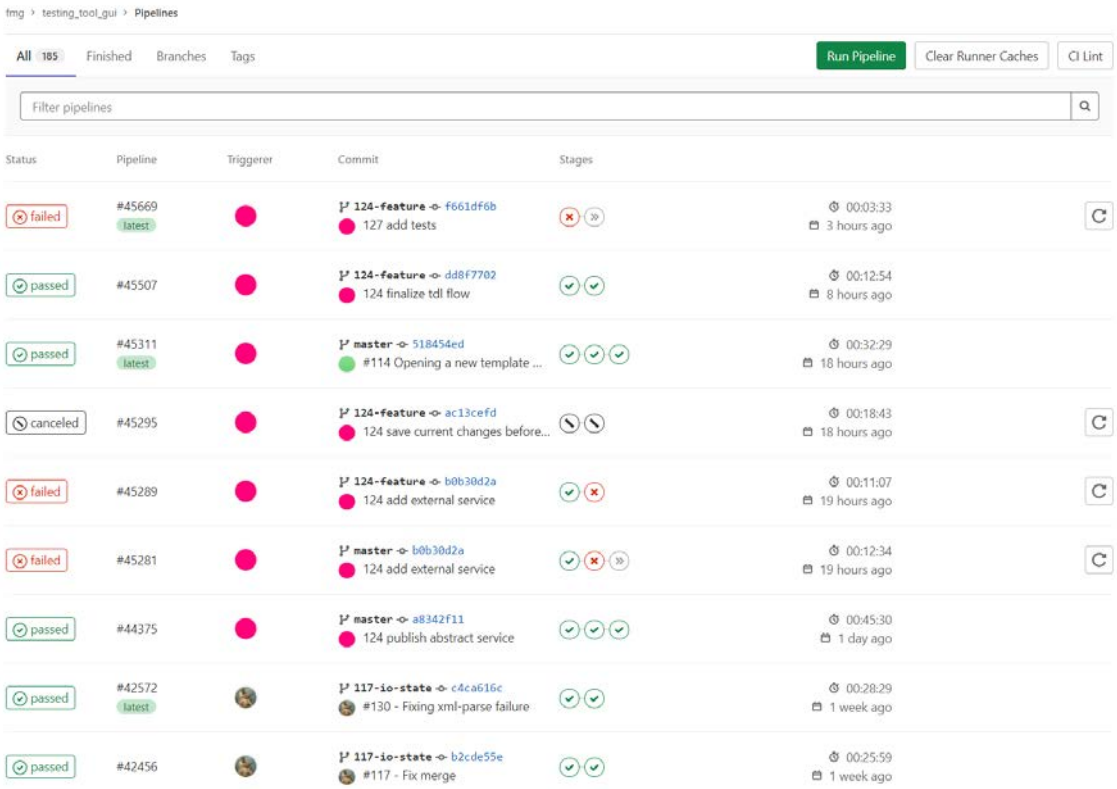
5.2 TDL-redaktor

TDL-redaktori täienduste raames lisati teenus nimega *External*. Läbi selle käib suhtlus välise teenusega, mis genereerib testitava süsteemimudeli ja TDL redaktoris loodud testi stsenaariumi baasil UPAALi faili, mis kirjeldab testi täitmise stsenaariumi.

Liidestust tehes tuli ilmsiks, et *Parser* teenuse testid on auklikud ja et kõik andmed, mis sisse võetakse, ei tule sealt enam samal kujul välja. Näiteks läksid olulised x ja y koordinaadid teatud elementide puhul kaduma või ei loetud mõningaid elemente üldse sisse. Seetõttu tuli teha *Parser* testide täiendus ja võtta ette teenuse parandus, et saaks liidestusele sisendiks anda päris sisendi, mis ei annaks veateateid. Kuna *Parser* teenust parandades tuli ilmsiks, et on vaja muuta mõnel määral projekti andmemudeli kuju, siis osutus see suuremaks ülesandeks kui oskasime oodata.

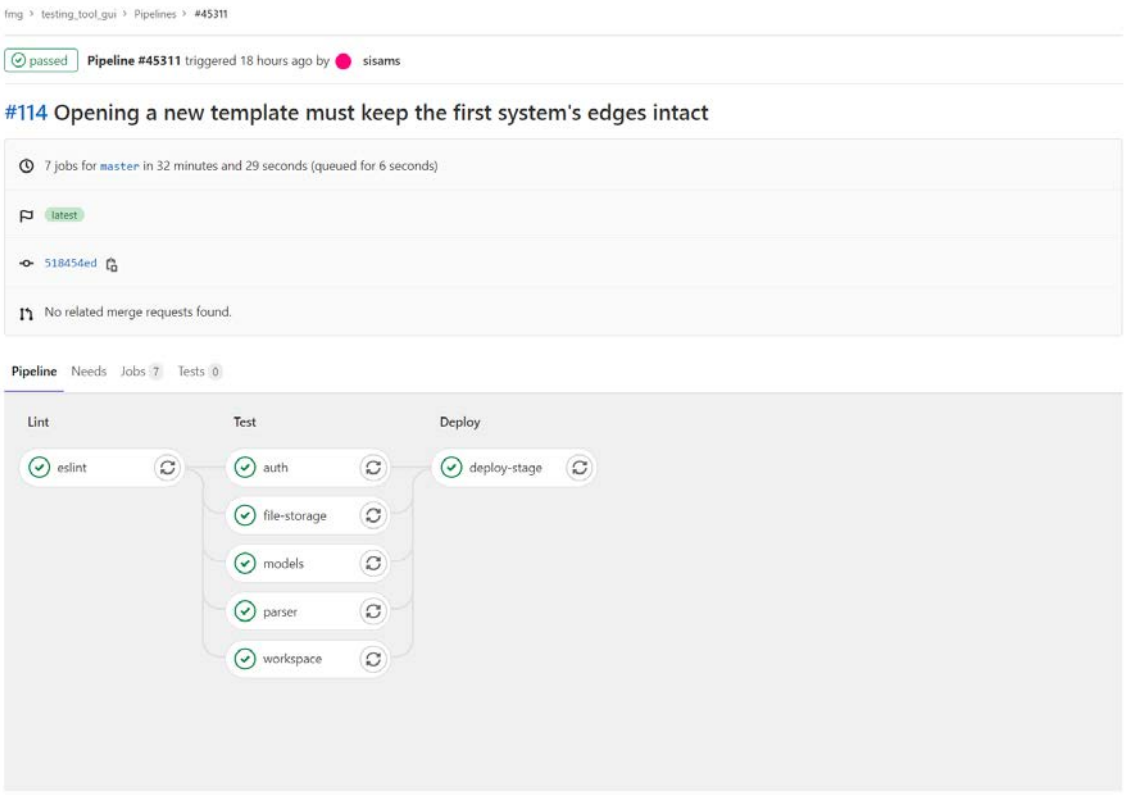
5.3 Töövoo muudatused

Varasemalt toimus koodistiili kontroll, testide jooksutamine ja rakenduse tarnimine manuaalselt. Et kiirendada arendusprotsessi ja saada kohest tagasisidet sai projekti käigus loodud CI/CD (Pidevkooste / pidevvalmidus) protsess. Protsess sai loodud projekti Gitlabi keskkonda, mis on nähtav Joonisel 7. Sellesse keskkonda on sisseehitatud võimekas CI/CD tööriist, mis võimaldab YAML (Inimloetav andmete serialiseerimise keel) faili baasil mugavalt protsesse ja samme defineerida.



Joonis 7. Projekti pidevkooste / pidevvalmiduse vaade projekti Gitlabis

Protsessil on hetkel 3 sammu. *Lint*, mille käigus kontrollitakse koodistiilist kinnipidamist ja süntaksivigade puudumist. *Test* kus jooksevad kõik testid ja genereeritakse testiraport. *Deploy* mis läheb *master* harus automaatselt käima ja paneb kõik eelnevad sammud edukalt läbinud rakenduse versiooni Tallinna Tehnika Ülikooli serverisse püsti, vt Joonis 8.



Joonis 8. Projekti pidevkooste / pidevvalmiduse sammud Gitlabi vaates

5.4 Tehniline võlg

Projektiga alustades tundus rakenduse tehniline lahendus ja seisund päris hea. Rakendust esmakordselt käima panna proovides ilmnes, et päris paljud teenused ei tööta nii nagu nad peaksid, sest paljud NPM (*Node Package Manager*) pakid olid liiga liberaalsete või puuduvate versiooni lukkudega. Et kogu lõputööks olev aeg ei kuluks versiooniuuenduste tõttu muutunud süntaksi muutmisele panime me pakkide versioonidele lukud külge ning projekti jooksutamiseks on kasutusel Node.js viimane pikaajalise toega versioon (14.17.0). Pakkide versioonid on lukustatud nii, et aktsepteeritakse turva vigade parandusi ja väiksemaid muudatusi.

Projektis oli kasutusel rakenduse koodi stiili ja süntaksivigade püüdmiseks mõeldud Eslint, aga graafilises rakenduses oli peaaegu kõigis vaadetes see välja lülitatud. Pärast Eslind-i täielikku lubamist tuli tegeleda üle tuhande veateatega, millest õnneks lõviosa lahenes linteri enda käe läbi. Järgi jäi ~500 veateadet, millega tuli käsitsi tegeleda. Väga palju oli kohti, kus kasutati samadele väärtustele viitamiseks nii *camel case*-i ja *snake case*-i. Põhjusele mõtlema jäädes tuli välja, et andmebaas, mille väljad kasutavad *camel*

case-i puudus ümber tõlkija meetod, mis muudaks andmebaasist tulevad ja sinna minevad objektid mõlema süsteemiga automaatselt ühilduvaks.

Teste jooksutades ilmnes, et kõikidest teenustest ainult ühe testid andsid positiivse tulemuse. Mis nõudsid parandamist. *Parser* teenuse test oli näiline, mis tegelikult midagi ei testinud, ega isegi funktsioneerinud. Testi parandades selgus, et *Parser* teenus ei tööta nii nagu peaks ja sisend ja väljund ei ole sama sisuga.

6 Valideerimine

Selles peatükis räägime kuidas toimub meie projektis valideerimine.

Testimises oleme lähenenud *superagiilselt* [8], mis näeb ette, et testimises otsitakse kuldset keskteed, kus maksimeeritakse regressiooni avastamise võimekus, aga minimeeritakse testide hooldusele kuluv aeg. Põhiliselt kasutame integratsiooniteste, millega kontrollime, et teenused funktsioneeriksid nagu nad peavad. Projekt on võrdlemisi algfaasis ja graafiline liides muutub pidevalt. Sellele teste kirjutades kuluks palju aega testide endi hooldamisele ja seetõttu otsustasime graafilist liidest käsitsi testida.

Suureks osaks valideerimisest on ka regulaarsed koodi ülevaatused, mis aitavad võimalikke murekohti leida enne kui need projekti jõuavad. Suurema funktsionaalsuse puhul viisime neid läbi Gitlabis *pull requesti* näol. Väiksemate osade puhul saime tiimi suuruse tõttu lihtsalt enda commiti kohta tagasisidet küsida.

7 Tulemused

Püstitatud eesmärkide saavutamiseks loodi kasutajaliides, mis võimaldab kasutajal vaadelda komponentide vahelisi sisendite ja väljundite sünkroniseerimistingimusi, täiendati olemasolevat kasutajaliidest TDL kompileerimise võimekusega ning lisati graafi elementide filtreerimise grupeerimise funktsionaalsus.

Lisaks loodi projektile pidevkooste ja pidevvalmiduse funktsionaalsus ning täiendati eelmise meeskonna poolt loodud kasutusjuhendit (vt Lisa 2.) uue funktsionaalsusega.

7.1 Pidevkooste ja pidevvalmidus

Pideva integratsiooni ja pideva juurutamise funktsionaalsus sai täielikult implementeeritud. Iga GitLabi salve lükatud muudatuse puhul lähevad tööle protsessid, mis kontrollivad koodi stiili, jooksutavad integratsiooni teste ning juurutavad tehtud muudatused kliendile kättesaadavasse serverisse.

7.2 I/O-vaade

I/O-vaates sai implementeeritud peaaegu kõik vajalik funktsionaalsus. Valmis sai mudelite lisamine ja mudelite vahelise sisendite ja väljundite sünkroniseerimis-kanalite visualiseerimine ning partitsioonide loomine. Lisaks tehti valmis ka I/O-vaate olekute talletamise funktsionaalsus, et hiljem oleks võimalik tööd sama koha pealt jätkata.

Implementeerimata jäi I/O-vaates kanalite dünaamiline genereerimine nii, et kanaleid ei peaks manuaalselt korrastama keerulisemate mudelite puhul, kanalitele vajutades kuvatavad muutujate nimekirjad ning asünkroonse suhtlemise tarvis globaalsete muutujate visualiseerimine.

7.3 Graafi redigeerimise vaade

Graafi redigeerimise vaates sai valmis kõik kavandatud funktsionaalsus. Implementeeriti graafi elementide filtreerimise grupeerimine ning lisati võimalus töötada kuni neljal töölaual samaaegselt.

8 Kommentaarid

Töö käigus ei saanud kogu funktsionaalsus implementeeritud kõige puhtamal ning intuitiivsemal viisil.

8.1 Graafi redigeerimise töölaud

Kui graafi redigeerimise töölauad on jaotatud mitmeks osaks, saab liigutada avatud mudelite kaarte poolituste vahel. See sai lahendatud viisil, mille puhul peab vajutama esmalt liigutatavale kaardile ning seejärel konkreetsele nupule poolitusel, millele soovitatakse kaardi asetada. Siinkohal oli soov lahendada selle funktsionaalsuse veebibrauseritest tuttavama ning intuitiivsema kaartide lohistamisega, kuid tingitult eelnevatest arendusvalikutest ei olnud selleks vajalik refaktoreerimine ajaliselt kuidagi õigustatud. Teeke nagu Vue.Draggable ei olnud võimalik mõistlikult kasutada põhjusel, et igas poolituses luuakse eraldi BootstrapVue kaartide komponent, mille skoobid jäävad üksteise omadest välja. Ka enda lahenduse käsitsi kirjutamine oleks siinkohal jätnud vähe aega teiste prioriteetsemate arenduste jaoks.

9 Järeldused ja edasised sammud

Projekt sai põhimahus täiendatud, kuid kui poleks pidanud paljusid vigasid otsima, parandama ja koodi refaktoreerima, oleks mahtunud ilmselt rohkem funktsionaalsust projekti skoopi. Kuna projektil on veel arenguruumi, siis on loodud grupitöö gitlabi salve wiki lehele kasulikud juhendid, et muuta järgmistele inimestele projekti ülevõtmine võimalikult valutuks. Et protsessi konkreetsemaks teha on pandud lukku kasutatud pakkide versioonid ning projekti jooksutamiseks on kasutusel kindel Node.js versioon pikaajalise toega.

Edasiste sammudena oleks järgmisel meeskonnal esiteks mõistlik vaadata GitLabi salve ülesannete backlogi. Seal on välja toodud praeguse meeskonna poolt leitud väiksemad probleemid, millega ei olnud aega tegeleda ning ülesanded on piltide ja kirjeldustega lahti seletatud. Väiksemate vigade lahendamine peaks andma järgmisele meeskonnale hea algse ülevaate projekti struktuurist.

Teiseks oleks hea mõte kirjutada kasutajaliidese teste, kasutades selle jaoks mingit veebirakenduste testimisraamistikku, näiteks Seleniumi. Hetkel toimub kõikide visuaalsete elementide testimine manuaalselt. Kui arendada juurde testid, mis katavad lihtsama funktsionaalsuse, oleks arendusprotsess töökindlam.

Lõpuks tooks välja selle, et vajalik funktsionaalsus on vaja arendajatele väga hästi selgeks teha. Projekti raames oli meeskonnal natukene keeruline aru saada näiteks sellest, et mis funktsionaalsust täpsemalt I/O-vaatest oodatakse. Kui minna edasi järgmise suurema tüki juurde antud rakenduses, milleks oleks simulatsiooni jooksutamise võimekus, siis on soovituslik väga täpselt paika panna kõik elemendid ja nende atribuudid, mis võivad kuidagi simulatsiooni jooksutamist mõjutada.

Kasutatud kirjandus

[1] Codecademy, *What is REST?*, [Online]. Loetud aadressil: https://www.codecademy.com/articles/what-is-rest Kasutatud 24.05.2021

[2] UPPAAL, *Introduction*, 2019, [Online]. Loetud aadressil: https://www.it.uu.se/research/group/darts/uppaal/about.shtml Kasutatud: 10.05.2021.

[3] J. Veispak, P. Roosiväli, H. Viirmäe, "Graafiline kasutajaliides ajaga automaatide põhiste testide modelleerimiseks" [Bakalaureusetöö], Tallinna Tehnikaülikool, Tallinn, Eesti, 2020. Kasutatud 10.05.2021

[4] Microservices, *Pattern: Monolithic Architecture*, [Online]. Loetud aadressil: https://microservices.io/patterns/monolithic.html Kasutatud 12.05.2021

[5] Microservices, *What are microservices?*, [Online]. Loetud aadressil: https://microservices.io/ Kasutatud 12.05.2021

[6] ECDAR, *Main Page*, [Online]. Loetud aadressil: http://people.cs.aau.dk/~adavid/ecdar/ Kasutatud 12.05.2021

[7] A. David, K. G. Larsen, A. Legay, U. Nyman, A. Wąsowski, "Timed I/O automata: a complete specification theory for real-time systems," Computer Science, Aalborg University, Denmark 2006 [Online], https://www.researchgate.net/publication/221422262_Timed_IO_automata_A_complete_specification_theory_for_real-time_systems Kasutatud 23.05.2021

[8] Concise, *What the hell is #superagile?*, 2019, [Online]. Loetud aadressil: https://concise.ee/blog/what-the-hell-is-superagile Kasutatud 14.05.2021

[9] G. Behrmann, A. David, K. G. Larsen, "*A Tutorial on Uppaal 4.0,*" Department of Computer Science, Aalborg University, Denmark 2006 [Online], https://www.it.uu.se/research/group/darts/papers/texts/new-tutorial.pdf Kasutatud 23.05.2021

[10] Bootstrap, *Main page*, [Online]. Loetud aadressil: https://getbootstrap.com/ Kasutatud 24.05.2021

[11] Jest, *Main page*, [Online]. Loetud aadressil: https://jestjs.io/ Kasutatud 24.05.2021

[12] P5, *home*, [Online]. Loetud aadressil: https://p5js.org/ Kasutatud 24.05.2021

[13] Vuex, *What is Vuex?*, [Online]. Loetud aadressil: https://vuex.vuejs.org/ Kasutatud 24.05.2021

[14] Koa, *Introduction*, [Online]. Loetud aadressil: https://koajs.com/#introduction Kasutatud 24.05.2021

[15] Github, *SuperTest*, 2021, [Online]. Loetud aadressil: https://github.com/visionmedia/supertest#about Kasutatud 24.05.2021

[16] UPPAAL, *VERIFYTA*, [Online]. Loetud aadressil: https://docs.uppaal.org/toolsandapi/verifyta/ Kasutatud 24.05.2021

[17] BootstrapVue, [Online]. Loetud aadressil: https://bootstrap-vue.org/ Kasutatud 25.05.2021

[18] Typescript, [Online]. Loetud aadressil: https://www.typescriptlang.org/ Kasutatud 25.05.2021

[19] Ecma-international, ECMAScript, [Online]. Loetud aadressil: https://www.ecma-international.org/technical-committees/tc39/ Kasutatud 25.05.2021

[20] IBM, *What is software testing?*, [Online]. Loetud aadressil: https://www.ibm.com/topics/software-testing Kasutatud 25.05.2021

[21] M. Krichen, "Model Based Testing for Real-Time Systems", [Doktoridissertatsioon], Albaha University, Albaha, Saudi Arabia, 2007, [Online], Loetud aadressilt: https://www.researchgate.net/publication/332413629_Model_Based_Testing_for_Real-Time_Systems, Kasutatud 25.05.2021

Lisa 1 – Lihtlitsents lõputöö reprodutseerimiseks ja lõputöö üldsusele kättesaadavaks tegemiseks[1]

Meie, Jürgo Pukk, Siim Sams, Jonas Tõnisson

1. Anname Tallinna Tehnikaülikoolile tasuta loa (lihtlitsentsi) enda loodud teose „Ajaga automaatide veebipõhine modelleerimiskeskkond“, mille juhendaja on Gert Kanter
   1.1. reprodutseerimiseks lõputöö säilitamise ja elektroonse avaldamise eesmärgil, sh Tallinna Tehnikaülikooli raamatukogu digikogusse lisamise eesmärgil kuni autoriõiguse kehtivuse tähtaja lõppemiseni;
   1.2. üldsusele kättesaadavaks tegemiseks Tallinna Tehnikaülikooli veebikeskkonna kaudu, sealhulgas Tallinna Tehnikaülikooli raamatukogu digikogu kaudu kuni autoriõiguse kehtivuse tähtaja lõppemiseni.
2. Oleme teadlikud, et käesoleva lihtlitsentsi punktis 1 nimetatud õigused jäävad alles ka autorile.
3. Kinnitame, et lihtlitsentsi andmisega ei rikuta teiste isikute intellektuaalomandi ega isikuandmete kaitse seadusest ning muudest õigusaktidest tulenevaid õigusi.

25.05.2021

1 Lihtlitsents ei kehti juurdepääsupiirangu kehtivuse ajal vastavalt üliõpilase taotlusele lõputööle juurdepääsupiirangu kehtestamiseks, mis on allkirjastatud teaduskonna dekaani poolt, välja arvatud ülikooli õigus lõputööd reprodutseerida üksnes säilitamise eesmärgil. Kui lõputöö on loonud kaks või enam isikut oma ühise loomingulise tegevusega ning lõputöö kaas- või ühisautor(id) ei ole andnud lõputööd kaitsvale üliõpilasele kindlaksmääratud tähtajaks nõusolekut lõputöö reprodutseerimiseks ja avalikustamiseks vastavalt lihtlitsentsi punktidele 1.1. ja 1.2, siis lihtlitsents nimetatud tähtaja jooksul ei kehti.

Lisa 2 - Kasutusjuhend

Rakenduse avamisel on kasutajal võimalik sisse logida. Kasutaja puudumisel tuleb kasutaja registreerida. Selleks tuleb navigeerida registreerimise vaatele. Registreerides suunatakse kasutaja tagasi sisselogimise vaatesse ning pärast sisse logimist jõuab kasutaja projekti vaatesse [3, lk 29].

Projekti vaates on võimalik teha erinevaid valikuid. Värskelt registreeritud kasutajal ei ole ühtegi valmis projekti. Projekti loomiseks on olemas navigatsiooni riba all kaks valikut "*Create a new project*" ja "*Create a new project with XML*". Valides "*Create a new project*" avaneb kasutajale modaal, kus on võimalik määrata oma projektile nimi ja kirjeldus. Valides "*Create a new project with XML*" on võimalik kasutajal importida oma olemasolev UPPAAL projekt veebirakendusse ning määrata projektile nimi ja kirjeldus. Projekti edukal loomisel lisandub loodud projekt avalehe vaatesse [3, lk 29].

Projekti ikoonile saab vajutada parema ja vasaku hiireklõpsuga. Parema klõpsuga ilmub vajutatud projekti juurde kiiremad redigeerimise võimalused. Projekti saab lisada külgribale, et seda oleks võimalik kiiremini avada ning vajadusel kustutada. Vasaku klõpsu juhul ilmub projekti detailvaade, kus on võimalik muuta projekti nime, kirjeldust ning lisada teisi kasutajaid selle projekti haldajaks või vaatajaks. Vajutades detailvaates kausta ikoonile on võimalik alla laadida projekti XML kujul, et seda vajadusel UPPAAL rakenduses redigeerida [3, lk 29].

Lisaks on detailvaates võimalik projektiga siduda erinevaid faile, näiteks PDF-formaadis failid, mis sisaldavad põhjalikumat kirjeldust projekti kohta [3, lk 29].

Projektidest on samuti võimalik teha koopia, et oleks võimalik testida suuremaid muudatusi, säilitades vana projekti mudeleid [3, lk 29].

Projekti avamisel navigeeritakse projekti vaatesse. Projekti vaates tuleb mudelite redigeerimiseks alguses teha süsteem. Kui süsteem on juba olemas, saab parema hiirenupuga süsteemi peale vajutades avada menüü ning automaadi mudelit on võimalik avada uuel töölaual. Avades mitu mudelit erinevatel töölaudadel tekib lõhestatud vaade,

kus on võimalik näha mitut mudelit korraga [3, lk 30]. Mudelite töölaudade vahel liigutamiseks tuleb esmalt vajutada avatud automaadi kaardil olevale nupule. Seejärel tuleb vajutada tahetud sihttöölaual ilmuvale plussmärgiga nupule. Automaadi mudeli loomiseks tuleb vajutada vasakul oleval vaheribal deklaratsiooni kirjale parema hiirenupuga ning valida "*Add new system*". Loodud mudelile uuesti parema hiirenupuga vajutades on võimalik ka süsteemi nime muuta.

Mudeli aknale parema hiirenupuga vajutades tekib võimalus luua graafile uus sõlm. Erinevaid sõlmi saab ühendada sõlmele hiire rullikuga vajutades ning liikudes hiirega teise peale ja vajutades vasakut hiire nuppu. Vajutades vasakut hiire nuppu enne sõlme, saab tekitada graafile vahepunkte. Graafil olevatele punktidele ja äärtele parema hiirenupuga vajutades ilmub rohkem võimalusi. On võimalik muuta äärte ja punktide värvi, neid saab kustutada ning on võimalik redigeerida nende atribuute [3, lk 30].

Vajutades redigeerimise nupule ilmub ääre ja punkti puhul eraldi modaal, kus saab redigeerida nende atribuute. Lisaks sellele, et punkte saab eraldi kustutada on võimalik vasakut klõpsu vajutada ja tõmmata üle pinnase, mis muudab aktiivseks mitu elementi korraga ning tagasilükkeklahvi vajutades kustutatakse valitud elemendid [3, lk 30].

Graafi redigeerimise vaates on võimalik CTRL ja Z klahvi kombinatsiooni abil muudatused tagasi võtta ning CTRL ja Y klahvikombinatsiooni abil muudatused tagasi tuua. Graafi mudeli aknas keskmist hiireklõpsu vajutades on võimalik kõiki pinnasel olevaid elemente korraga liigutada [3, lk 30].

Vasakul külje ribal on näha *Trapset*-menüüd. Sinna saab süsteemi nimele parema hiirenupuga vajutades luua uue *Trapset*-i. Pärast *Trapset*-i loomist on võimalik sama mudelil olevate kaarte peale paremat klõpsu vajutades lisada kaar *Trapset*-i [3, lk 30].

Mudeli redigeerimise vaates on võimalik vasakust küljeribast lülitada välja nii kaarte kui ka sõlmede nimesid, kommentaare ja atribuute [3, lk 31]. Kui kasutaja soovib grupeerida erinevaid elemente mida soovib mudeli pealt välja lülitada, on võimalik külje menüüst "*View declarations*" peale parema klõpsuga vajutades lisada uus vaade. Vaate muutmisel on avaneb kasutajale modaal, kus on võimalik defineerida vaatele nimi, kirjeldus ja valida mudeli elemendid mida soovitakse filtreerida. Pärast vaate salvestust saab kasutaja külje menüüs defineeritud vaadet sisse ja välja lülitada.

I/O-vaade on avatav ülemisel ribal laiuvast *Views*-rippmenüüst valides sellelt alampunkti I/O. Vajutades automaatide loetelus mõnele automaadile parema klõpsuga ning valides *Open* avatakse vastava automaadi mudel töölaual koos sisendite ja väljunditega, mis on defineeritud redigeerimise vaate *Sync* atribuudiga. Valides töölaua menüüst partitsioonide tööriist, on võimalik töölaual tühjale kohale vajutades ning hiirt lohistades joonistada partitsioon. Partitsiooni üleval vasakus nurgas selle nimele vajutades on võimalik ka nime muuta. Loodud elemendid on töölaual hiirega lohistades liigutatavad ning mudelitevahelistele kanalitele on võimalik nendele hiire rullikuga vajutades luua liigutatavad vahepunktid ehk naelad.

TDL redigeerimise vaade avaneb ülemises rippmenüüs *TDL and tester* alampunktile *TDL editor* vajutades. Vaate vasakul küljel asub *Trapset*-i tulp, kuhu ilmuvad eelnevalt loodud *Trapset*-id, mida saab enda TDL-lausete koostamisel kasutada. Täites ära kõik väljad on võimalik TDL-väärtust salvestada ja kompileerida. Paremale küljele ilmuvad kõik juba loodud TDL-lausete nimed, millele vajutades saab seda lauset muuta või kustutada. Vaate üleval paremal nurgas asub nupp nimega "*New Expression*" millele vajutades luuakse uus TDL-lause, mida on seejärel võimalik täita.