TARTU ÜLIKOOL
Arvutiteaduse instituut
Informaatika õppekava

Kaspar Kadalipp

LTL-põhise vastavuskontrolli integreerimine protsessikaeve rakendusse RuM

Bakalaureusetöö (9 EAP)

Juhendaja(d): Fabrizio Maria Maggi
Anti Alman

Tartu 2021

LTL-põhise vastavuskontrolli integreerimine protsessikaeve rakendusse RuM

Lühikokkuvõte:

Bakalaureuse töö käigus täiendati protsessikaeve rakenduse RuM funktsionaalsust. RuM on deklaratiivse protsessikaeve metoodikaid kasutav raamistik. Lisatav funktsionaalsus on LTL keelel põhinev vastavuskontrolli metoodika, mis võimaldab analüüsida oluliselt keerukamaid protsesse, kui olemasolev vastavuskontroll. Vastavuskontrolli põhimõte on protsessi eeldatava ja tegeliku käitumise võrdlemine ning sellest saadud info põhjal protsessi parendamine.

Soovitava funktsionaalsuse saavutamiseks integreeriti antud töös rakendusse RuM rakenduse ProM pistikmoodul LTL Checker. Lisatud pistikmooduli kujundust tuli muuta, et see sobituks olemasoleva rakenduse kujundusega. Prototüübi loomisel kasutati kujundusplatvormi Adobe XD. Töös on kirjeldatud kasutajaliidese kujundamise protsessi ning tehtud algust reaalse implementatsiooniga. Tulemusena muutub pistikmooduli funktsionaalsuse kasutamine rohkem võimalusi pakkuvaks ning kasutajale mugavamaks.

Võtmesõnad:

Protessikaeve, protsessianalüütika tööriist, LTL Checker, vastavuskontroll, kasutajaliidese disain

CERCS: P170 - Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Integrating LTL-based conformance checking into the process mining application RuM

Abstract:

In the course of this Batchelor's thesis new features were added to the desktop application RuM. RuM contains a comprehensive set of declarative process mining tools. Added functionality provides an additional LTL-based conformance checking technique which enables the analysis of more complex processes than what is possible with conformance checking techniques currently available in RuM. The main aim of conformance checking is to detect deviations between the modelled behaviour and the observed behaviour of the process and to use that information as a basis for process enhancement.

To achieve the desired functionality, the ProM plug module LTL Checker was integrated into the RuM application. The design of the added plugin had to be changed so that it could be compatible with the user interface of the existing application. The design tool Adobe XD was used to create a prototype. The thesis describes the process of creating a user interface and implementing part of the desired functionality. As a result, using the LTL Checker will become more convenient and will proovide users with additional features.

Keywords:

Process mining, process analytics tool, LTL Checker, Conformance checking, User Interface Design

CERCS: P170 - Computer Science, Numerical Analysis, Systems, Control

Sisukord

Sissejuhatus

Käesoleva rakendusliku lõputöö eesmärk on täiendada protsessikaeve rakenduse RuM [1,2] funktsionaalsust ning disainida lisatavale funktsionaalsusele vastav kasutajaliidese osa. RuM sisaldab mitmeid erinevaid deklaratiivse protsessikaeve algoritme ja on valminud Tartu Ülikooli ning mitmete teiste ülikoolide koostöös. Rakenduse kasutajaskond on valdavalt teadlased ja valdkonnaeksperdid. Rakenduse arendusel on olnud oluline, et kasutajaliides oleks intuitiivne ja ei nõuaks rakenduses põhiliselt kasutatava Declare keele põhjalikku mõistmist. See on omakorda muutnud rakenduse otstarbekaks ka protsessikaeve valdkonnas vähemkogenud kasutajatele.

RuM on töökorras ja juba kasutuses, kuid mõningate keerukamate äriprotsesside ja ärireeglite analüüsiks võib olla vaja täiendavat funktsionaalsust. Üks selline funktsionaalsus on LTL keelel põhineva vastavuskontrolli teostamine. Rakenduses olemasolev vastavuskontroll kasutab deklaratiivse keele Declare mudeli eeskirju (*constraint*), mis küll tuginevad LTL keelele, aga sellegipoolest ei oma niivõrd tugevat väljendusvõimekust kui LTL keel ise. Käesoleva lõputöö eesmärk on võimaldada ka LTL keeles väljendatud mudeli eeskirjade põhjal vastavuskontrolli teostamine. See funktsionaalsus on suunatud kogenud kasutajatele, kellel võib tekkida vajadus kontrollida sellist protsessi käitumist, mida on keeruline või isegi võimatu väljendada ainult Declare keelt kasutades.

Lisaks vastavuskontrollile saaks selle tööriistaga ka logifaili filtreerida, eraldades alamosa, mis eeskirju rahuldab. Logifailis on väga palju infot ja filteerimine kiirendaks oluliselt ülevaate saamist, sest filtreerimisel eemaldatakse ebaoluline olulisest. Alamosa põhjal koostatud töövoo skeem on palju väiksem ning on seetõttu inimesele märgatavalt arusaadavam.

Hetkel kasutatakse LTL keelel põhineva vastavuskontrolli teostamiseks põhiliselt protsessikaeve rakendust ProM [3]. ProM võimaldab kasutada suurt hulka protsessikaeve algoritme, aga selle rakenduse kasutamine on kohmakas ning kasutajaliides aegunud. Samuti on ebamugav töötada paralleelselt mitme erineva rakendusega. Sellest tulenevalt luuakse rakenduses RuM uus sektsioon, millesse integreeritake rakenduse ProM pistikmoodul LTL Checker, mis võimaldab kasutada LTL keeles väljendatud mudeli eeskirju vastavuskontrolli teostamiseks.

Bakalaureusetöö koosneb kolmest osast. Töö esimene osa annab ülevaate protsessikaevest, keskendudes põhiliselt vastavuskontrollile, ja kirjeldab LTL-põhise vastavuskontrolli jaoks vajalikke sisendeid ning vastavuskontrolli võimalusi protsessikaeve rakendustes ProM ja

RuM. Teine osa kirjeldab kasutajaliidese loomise protsessi, kasutatatud töövahendeid, disaini eesmärke ja lõpliku kujundust. Kolmas osa tutvustab rakenduse RuM tehnilist üleshitust ning implementeeritud funktsionaalsust.

1. Taust

See peatükk annab ülevaade protsessikaeve olemusest keskendudes eelkõige vastavuskontrollile. Sellele järgneb LTL valemite kirjeldus protsessi reeglite kontekstis ning sündmuslogi formaadi XES (*eXtensible Event Stream*) kirjeldus. Viimased kaks alampeatükki annavad ülevaate rakenduses RuM olemasolevast vastavuskontrolli funktsionaalsusest ja ProM pistikmoodulist LTL Checker.

1.1. Vastavuskontroll

Nagu allikad [4–6] kirjeldavad, hõlmab protsessikaeve mitmeid erinevaid andmeanalüüsi metoodikaid, mis võimaldavad protsessi reaalset töökäiku tuvastada, modelleerida ning täiustada. Protsessikaeve kuuldub äriprotsesside juhtimise valdkonda (*business process management*) ning jaguneb kolmeks põhiliseks haruks: protsessi tuvastus (*discovery*), vastavuskontroll (*conformance checking*) ja protsessi täiustamine (*enhancement*). Käesolev töö keskendub vastavuskontrolli alla kuuluva funktsionaalsuse implementeerimisele ja sellest tulenevalt ei ole teised protsessikaeve harud käesolevas töös täpsemalt kireldatud.

Tarkvara kasutamisel on oluline teada, kas tarkvara töötab nagu ette nähtud. Levinud praktika selle jälgimiseks on salvestada protsesside iga vahesammu toimimise kohta info logifaili. Kui rakendus peaks mingil põhjusel vigasesse seisu jõudma ja selle tõttu ei toimi nagu ette nähtud, siis on oluline tuvastada vigasesse seisu jõudmise põhjused. Kirjete olemasolu vigadest (ja vigadele eelnenud vahesammudest) teeb vigaste kohtade tuvastamise ja parandamise arendajate jaoks kergemaks.

Vigade tuvastamisel on selline lähenemine tõhus, aga tihtipeale ei ole see piisav olukorras, kus tarkvara või tarkvara poolt toetatud äriprotsessi töövoog ei toimi nagu peaks või miski põhjustab põhjendamatut viivitust. Äriprotsesside töötamine on rakendustes sageli peidetud taustale, mis omakorda raskendab probleemsete kohtade tuvastamist. Logifailides on üldjuhul suur hulk andmeid, mis võivad olla inimesele arusaadaval kujul, ent mahu tõttu ei suudeta neid andmeid tervikuna siiski manuaalselt analüüsida. Sellise olukorra lahendamisel on kasu vastavuskontrollist, mis võimaldab probleemsete töövoogude automaatsemat tuvastamist ning põhjalikumat analüüsi. Näiteks on võimalik kindlaks teha konkreetsed juhused, kus protsessi kasutajad jätavad mingid olulised tegevused tegemata, teevad tegevusi vales järjekorras või eiravad muid protsessi spetsiifilisi reegleid.

Vastavuskontrolli sisendiks on sündmuslogi ja protsessi töökäigu mudel (*process map*). Mudel võib olla nii olemasoleva sündmuslogi põhjal automaatselt tuvastatud kui ka käsitsi

koostatud. Mõlemal juhul on tegu vastavuskontrolli jaoks sobiva sisendiga. Vastavuskontroll tugineb automatiseeritud algoritmidele, mis võrdlevad sündmuslogi sisu ja protsessi töökäigu mudelit. Võrdluse tulemuseks on üldjuhul detailne ülevaade mudeli ja sündmuslogi erisustest. Seeläbi on võimalik tuvastada protsessi töötamise kitsaskohad ja kõrvalekalded, mis võivad viidata nii probleemidele mudelis (näiteks kasutajad on leidnud efektiivsema viisi protsessi eesmärkide täitmiseks) kui ka probleemidele selles, kuidas protsessi reaalsuses täidetakse (näiteks mõni seaduse poolt nõutud tegevus jääb mõningatel juhtudel tegemata). Vastavuskontrolli metoodikat saab rakendada mistahes äriprotsessile, eeldusel et leiduvad protsessi kirjeldavad andmed.

Käesolevas töös käsitletavad protsessi töökäigu mudelid põhinevad LTL valemitel (täpsemalt kirjeldatud järgmises peatükis), mis võimaldavad kirjeldada protsessi töövoogu deklaratiivselt, nii saab defineerida matemaatilisel loogikal põhinevaid eeskirju, mis kirjeldavad mingit osa protsessi käitumisest. Deklaratiivse lähenemise põhiline eripära seisneb selles, et protsessi kohta kirjeldatakse ainult olulised käitumise osad (näiteks millised tegevused peavad kindlasti toimuma) ja samas mitte-olulied protsessi osad „jäetakse vabaks“. Ehk protsessi täitmisel on lubatud kõik töövood, mis ei ole vastuolus protsessi eeskirjadega.

Rakenduses RuM on hetkel kasutusel Declare keelel põhinev vastavuskontroll. Declare põhineb LTL keelel ning on samuti deklaratiivne modelleerimiskeel, mis kasutab mudeli kirjeldamiseks eeskirju [7]. Keeles Declare on kasutada piiratud arv selge semantilise tähendusega eeskirju, mis teeb selle kasutamise lihtsaks võrreldes LTL keelega. See-eest LTL keel on palju väljendusrikkam, võimaldades kirjeldada ka sellist protsessi käitumist, mida ainult Declare keele eeskirju kasutades oleks keeruline või võimatu teostada [8]. Näiteks protsess mis sisaldab kahte sellist tegevust, mille sooritajaks ei tohi olla sama isik. Selline olukord tekiks näiteks müügilepingu allkirjastamisel, kus üks isik ei saa mõlemat poolt lepingust ise allkirjastada.

1.2. Protsessi spetsifikatsioon LTL valemitega

Järgnevates lõikudes esitatud info põhineb LTL Checker käsiraamatul [9], kui pole näidatud teisiti.

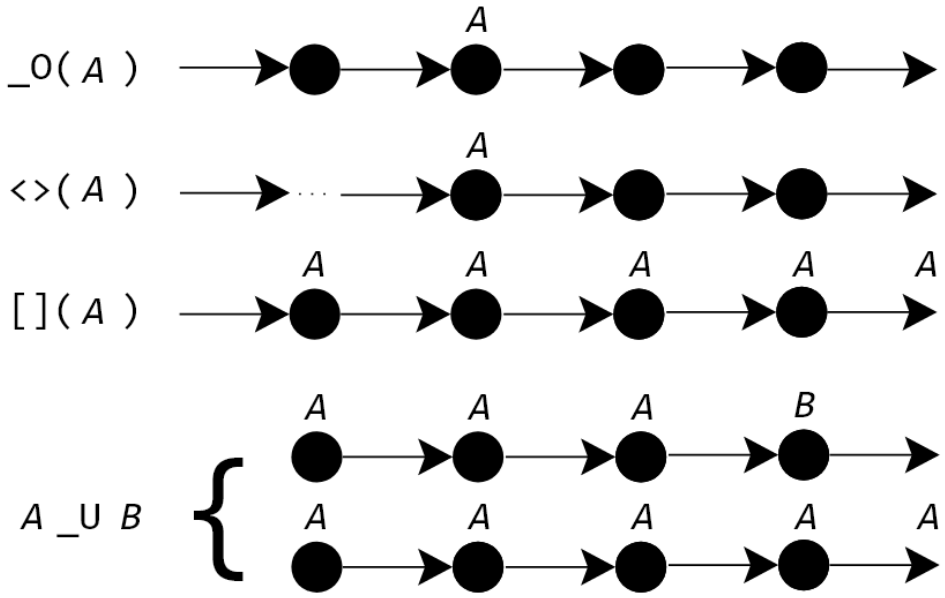
LTL keele valemit saab vaadata kui lausearvutuse valemit. Komponentlause moodustab mingi atribuudi väärtuse võrdlemisel kas literaaliga või mõne muu atribuudiga, mille tulemusena omandab komponentlause tõeväärtuse (tõene või väär). Järgnevates seletustes tähistavad *A* ja *B* mingit komponentlauset.

Võrdlemisel on kasutada operatsioonid: `==`, `>=`, `<=`, `!=`, `>`, `<`, `in`. Võrdus- ja võrldusmärkide kasutus on endastmõistetav. Operatsiooni `in` kasutatakse võrdlemisel hulgaga ning see kontrollib, kas atribuudi väärtus sisaldub antud hulgas. Oluline on ka lisada, et võrdlemisel peab vasakul pool alati olema atribuut ning kui antud nimetusega atribuuti sündmuse kohta ei eksisteeri, siis on võrdluse tulemus alati väär. Komponentlause kombineerimiseks on kasutada operatsioonid: `/\`, `\/`, `->`, `<->`, `!`. Nende operatsioonide kasutamine on esitatud joonisel 1, kus tähistusele *t* ja *v* vastavad tõeväärtused tõene ja väär.

A	B	$A\,/\backslash\,B$	$A\backslash/B$	$A\text{->}B$	$A\text{<->}B$	$!\,(A)$
t	t	t	t	t	t	v
t	v	v	t	v	v	v
v	t	v	t	t	v	t
v	v	v	v	t	t	t

Joonis 1: Lausearvutuse operatsioonide tõeväärtused (*propositional logic*)

Veel on kasutada ka ajalise loogika operatsioonid `_O`(*A*), `<>`(*A*), `[]`(*A*), *A* `_U` *B*. Need võimaldavad kontrollitavale sündmusele järgnevate sündmuste omaduste kontrollimist. Esimene operatsioon _O(*A*), nimetusega *next time*, kontrollib A kehtivust vaadeldavale sündmusele järgnevas sündmuses. Teine operatsioon <>(*A*), nimetusega *eventually*, kontrollib, kas *A* kehtib vähemalt ühe korra sündmuste jadas. Kolmas operatsioon `[]`(*A*), nimetusega *always*, kontrollib, kas *A* kehtib jada igal sündmusel. Neljas operatsioon *A* `_U` *B*, nimetusega *until*, kontrollib, kas *A* kehtib kuni hetkeni, millal *B* hakkab kehtima ning kui *B* kunagi ei kehti, siis peab *A* igal sündmusel kehtima. Joonisel 2 on illustratsioon erinevate operatsioonide kasutamisest. Joonisel iga rida on sündmuste jada, rea alguses on sündmuste jadale vastav operatsioon, jada esimene must täidetud ring on hetkel vaadeldav sündmus. Sümbolid *A* või B sündmuse peal tähistavad komponentlause kehtivust vastavas sündmuses. Kui sündmus ei ole tähega tähistatud, siis sinna sobib iga suvaline sündmus.



Joonis 2: Ajalised loogika operatsioonid (*temporal logic*)

LTL valemi operaatsioone on veel mitmeid. See peatükk piirdub operatsioonide kirjeldamisega, mida on võimalik kasutada ProM pistikmoodulis LTL Checker.

Lisaks käsitleme siin ka kahte LTL Checkeri spetsiifilist kvantor-operatsiooni (*quantification logic*), mille tähistus on valemites 1 ja 2. Nende funktsionaalsus on sarnane operaatoritega <>(*A*) ja `[]`(*A*).

(1) `forall[` <lokaalne muutuja> `:` <atribuut> `|` <valem> `]`

(2) `exists[` <lokaalne muutuja> `:` <atribuut> `|` <valem> `]`

Mõlema puhul valitakse atribuut, mille kõigi unikaalsete väärtustega sündmuste logis kontrollitakse järgneva valemi kehtivust. Seega enne püstkriipsu defineeritud lokaalne muutuja saab valemis pärast iga valemi kontrolli iteratsiooni uue väärtuse. Operatsioon `forall` on tõene, kui valem kehtib iga atribuudi väärtuse korral. Operatsioon `exists` on tõene, kui valem kehtib vähemalt ühe atribuudi vääruse korral. Täpsem operatsioonide matemaatiline esitus on kirjeldatud Aalst, Beer & Dongen aruandes [10].

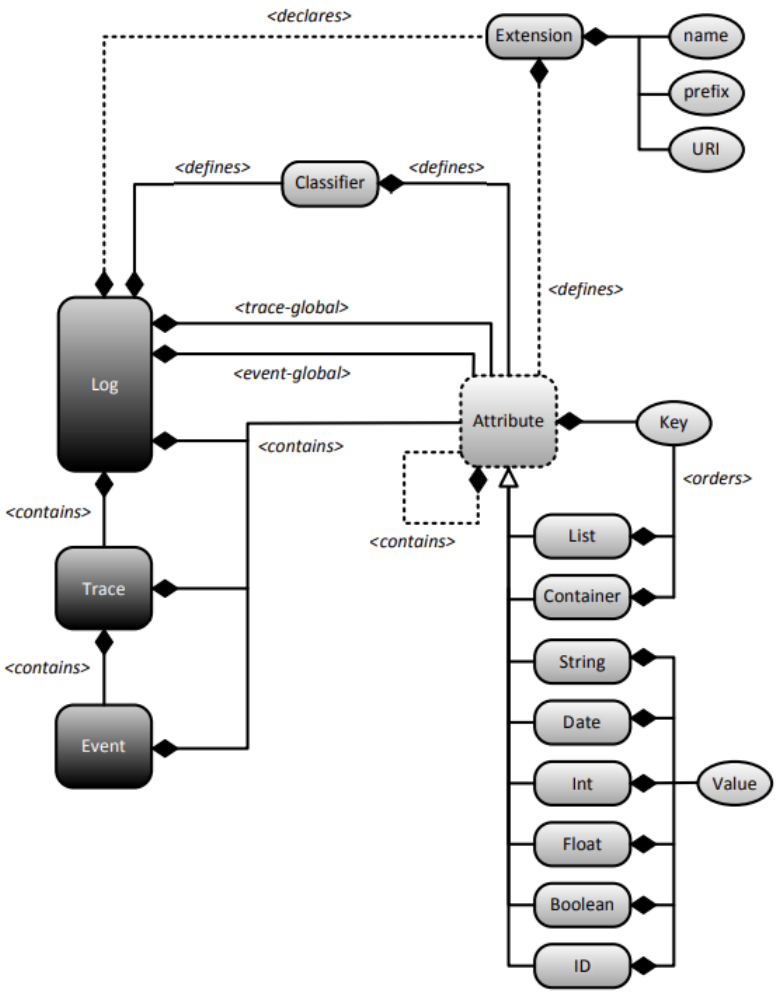
1.3. Sündmuslogi ja XES-formaat

Nagu protsessikaeve manifest [5] kirjeldab on protsessikaeve erinevate metoodikate rakendamise põhiliseks sisendiks sündmuste logikirjed. Samas reaalsetes infosüsteemides olemasolevad andmed ei ole sageli sobilikus formaadis, et neid saaks protsessikaeves muutmata kujul kasutada. Andmed võivad olla pärilt väga erinevatest allikatest: andmebaasi päring,

sõnumivahetuse ajalugu, tehingute logi, rakendusserveri logi jne. Seega on esmalt vaja eraldada vajalik info kogutud andmetest.

Protsessikaeveks vajalik info koosneb sündmustest (*event*) mis vastavad protsessi kirjelduses ette nähtud tegevustele (*activity*). Tegevused omakorda grupeeritakse mingi tunnuse alusel logijälgedeks (*trace*). Tegevuste gupeerimise tunnuseks on üldjuhul protsessi instantsi tunnus, milleks sobib sõltuvalt protsessist näiteks sessiooni identifikaator, lepingu number, ostukorvi identifikaator vms. Kõik selliselt eraldatud info koos moodustab sündmuslogi (*event log*), mille kvaliteet sõltub suurel määral algandmete kvaliteedist. Kui algandmed ei ole piisavalt kvaliteetsed, siis ei pruugi nendest andmetest eraldatud sündmuslogi anda protsessikaeve täpseid tulemusi ning võib mõningatel juhtudel viia isegi eksitavate järeldusteni.

Üks enim kasutatud formaate sündmuslogi salvestamiseks on XES (*eXtensible Event Stream*). XES on hierarhiline andmeformaat, mis tugineb XML (*Extensible Markup Language*) andmeformaadil. XES-formaat on loodud eesmärgiga tekitada standardne formaat sündmuste andmete vahendamiseks erinevate infosüsteemide vahel [11]. XES-formaadi põhiline kasutusala on protsessikaeve, aga see formaat sobib ka andmekaeve, tekstikaeve ja statistilise analüüsi jaoks [12].



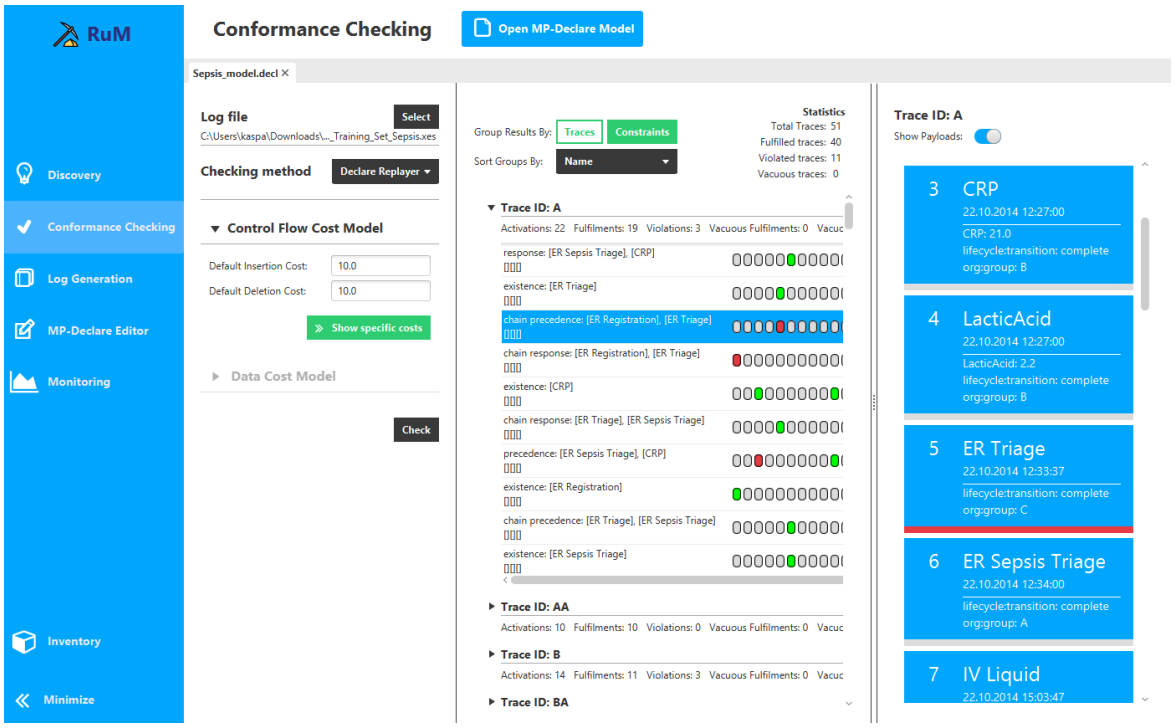
Joonis 3: XES standardi metamudeli UML 2.0 skeem [12]

XES formaadis fail sisaldab endas sündmuslogi ning on ka inimloetav, mis on olnud oluline printsiip formaadi väljatöötamisel [12]. Formaadi süntaks on toodud joonisel 3. Nagu jooniselt 3 näha, sisaldab logi (*log*) endas infot protsessi instantsi jooksul toimunud sündmusete (*event*) kohta, mis on logijälje (*trace*) kaupa grupeeritud. Ehk teisisõnu, sündmuslogi sisaldab infot protsessi instantsides toimunud vahesammude kohta protsessi instantside alustamisest kuni lõpetamiseni.

On oluline, et sündmused oleksid logijälje sees järjestatud nende toimumise järjekorras. Samuti peab igal sündmusel olema nimetus või mõni muu identifitseeriv tunnus. Kuigi sündmus peab omama vaid nime, on üldjuhul kasulik lisada juurde võimalikult palju täpsustavat infot ehk atribuute, näiteks ajatempel ja elütsükli seisund (aga ka protsessi spetsiifilised atribuudid nagu näiteks lepingu sõlmimise kuupäev, ostukorvi maksumus jms). Atribuudid, mis peavad protsessil ja sündmusel juures olema, on määratud märgendi *global* sees.

1.4. Protsessikaeve rakendus RuM

Nagu Alman, Ciccio, Haas, Maggi & Nolte [1,2] kirjeldavad on RuM protsessikaeve raamistik, mis võimaldab kasutada mitmeid deklaratiivse protsessikaeve metoodikaid. Rakenduse funktsionaalsus on jaotatud hetkel viieks sektsiooniks. Sektsioonis *Discovery* saab sündmuslogi põhjal tuvastada vastava Declare keeles protsessimudeli. Sektsioonis *Conformance Checking* saab kasutada Declare keelel põhinevat vastavuskontrolli. Sektsioonis *Log Generation* saab Declare keeles kirjeldatud protsessimudeli põhjal genereerida vastava kunstliku sündmuslogi. Sektsioon *MP-Declare Editor* võimaldab luua uusi ning muuta olemasolevaid Declare keeles kirjeldatud protsessimudeleid. Ning sektsioonis *Monitoring* saab näha animeeritud visualisatsiooni protsessi kulgemisest.



Joonis 4: *Conformance Checking* sektsioon rakenduses RuM

Käesoleva lõputöö fookus on vastavuskontrollil. RuM rakenduses olemasolevas vastavuskontrolli sektsioonis (joonis 4) on võimalik kasutada tööriistu *Declare Analyser* [13], *Declare Replayer* [14] ja *DataAware Declare Replayer*. Kõik nimetatud tööriistad kasutavad sisendina Declare keele eeskirju (laiendiga .decl fail) ja sündmuslogi. LTL keele valemeid sisendina kasutada ei saa.

Vasakul oleval külgribal saab muuta vastavuskontrolli parameetreid. Ning paremeetrid, mida valitud tööriistaga kasutada ei saa on hallid ja mitteaktiivsed. Peale vastavuskontrolli sooritamist kuvatakse paremal pool nimekiri logijälgedest koos vastavuskontrolli

tulemusega iga logijälje kohta. Logijälje peale vajutades saab näha ka täpsemat infot mudelis sisalduvate eeskirjade täitmise kohta. Järgnevalt eeskirja peale vajutades tekib paremal nimekiri sündmustest antud logijäljes. Sõltuvalt kasutatud tööriistast kuvatakse sündmuste kohta kas eeskirjadele vastavuse/mittevastavuse (*fulfillment/violation*) info või logijälje joondus (*trace alignment*), mille põhjal on võimalik aru saada, millised sündmused olid logijäljest puudu ja millised sündmused olid logijäljes üleliigsed.

1.5. Pistikmoodul LTL Checker rakenduses ProM

Nagu erinevates allikates [15,16] kirjeldatakse, on ProM (*Process Mining framework*) avatud lähtekoodiga modulaarne protsessikaeve raamistik. See sisaldab endas laialdaselt protsessikaeve funktsionaalsust pistikmoodulitena (*plugin*). Iga pistikmoodul kujutab endast üldjuhul mingi algoritmi implementatsiooni aga esineb ka lihtsamaid pistikmooduleid, mille eesmärk on näiteks logijälje visualiseerimine ilma logijälje sisu automaatse analüüsita.

Rakenduse ProM modulaarne arhitektuur võimaldab vähese vaevaga lisada enda tehtud pistikmooduleid. Paljude pistikmoodulite olemasolu teeb kasutajale enda tehtud mooduli arendamise ja lisamise lihtsamaks kuna olemasolevad pistikmoodulid on üldjuhul avatud lähtekoodiga. Seeläbi on ProM kujunenud suurepäraseks rakenduseks, millega uusi alogritme katsetada. Rakenduse probleemiks on olemasolevate pistikmoodulite erinevus, mille kasutamise erinev loogika vähendab kasutamise intuitiivsust ning muudab uutele kasutajatele rakenduse keerulisemaks. Sama võib öelda ka rakenduse ProM kasutajaliidese kohta.

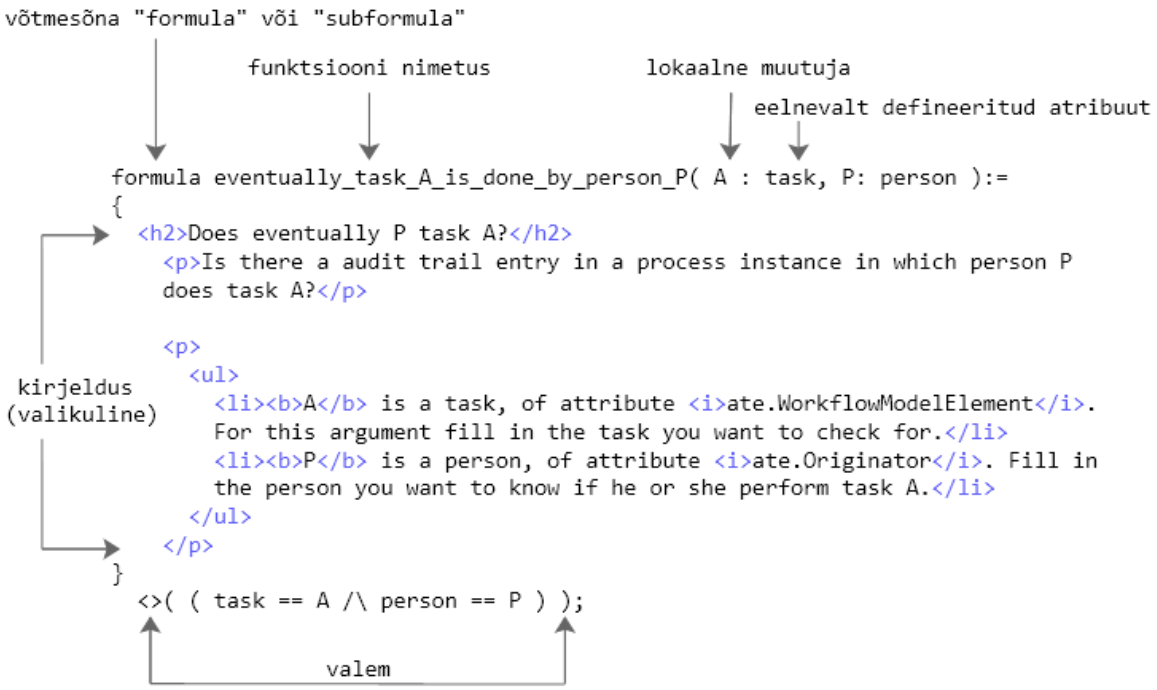
LTL Checker [17] on rakenduse ProM raamistikule loodud pistikmoodul, mis võimaldab sündmuslogile rakendada LTL valemite loogikat, et tuvastada LTL keeles väljendatud reeglitele vastavaid ja mitte-vastavaid logijälgi. Pistikmooduli kasutamiseks on algselt vaja see alla laadida, kasutades selle jaoks rakenduse ProM paigaldusega kaasaskäivat rakendust ProM Package Manager. Hetkel on saadaval LTL Checker versioon 6.9.108.

Pärast paigaldamist on ProM tegevused (*action*) vaates kasutada valikud LTL Checker ja LTL Checker Default. Pistikmooduli LTL Cheker kasutamiseks on sisendina vaja LTL valemite faili (laiendiga .ltl) ja sündmuslogi. LTL Checker Default sisaldab eeldefineeriud valemite malle ning seetõttu vajab sisendina ainult sündmuslogi.

Kahjuks eeldavad LTL Checker ja LTL Checker Default, et sündmuslogi on salvestatud MXML formaadis, mis on formaadi XES eelkäija ning pole enam kasutuses. Kuigi XES formaadis sündmuslogi faile on võimalik teisendada MXML formaati, siis see on kasutajatele tülikas ja seega vähendab vastavate ProM pistikmoodulite kasutatavust.

Kahjuks eeldavad LTL Checker ja LTL Checker Default, et sündmuslogi on salvestatud MXML formaadis, mis on formaadi XES eelkäija ning pole enam kasutuses [10]. Kuigi XES formaadis sündmuslogi faile on võimalik teisendada MXML formaati, siis see teisendus tuleb sagely teha eraldi tööriistaga ja on kasutajatele seega tülikas ning vähendab vastavate ProM pistikmoodulite kasutatavust.

Nagu pistikmooduli LTL Checker käsiraamat [9] kirjeldab, on LTL valemite failis kõigepealt muutujatena defineeritud sündmuslogis kasutatavad atribuudid ning igale muutujale on määratud ka literaali tüüp: number, hulk, sõne või kuupäev. Keel toetab ka regulaaravaldiste kasutamist sõne sees ning muutujaid saab vajadusel ümber nimetada. Nimetus *ate* (*audit trail entry*) vastab protsessi sündmuste atribuutidele ning *pi* (*process instance*) protsessi grupeeringu ehk logijälje atribuutidele. Seejärel on defineeritud funktsioonid ning abifunktsioonid. Funktsiooni osad on kirjeldatud joonisel 5. Valemi sees saab kasutada abifunktsiooni tulemust, seega asendades valemi korduvad osad abifunktsiooni väljakutsega saab valemit teha lühemaks ja loetavamaks.



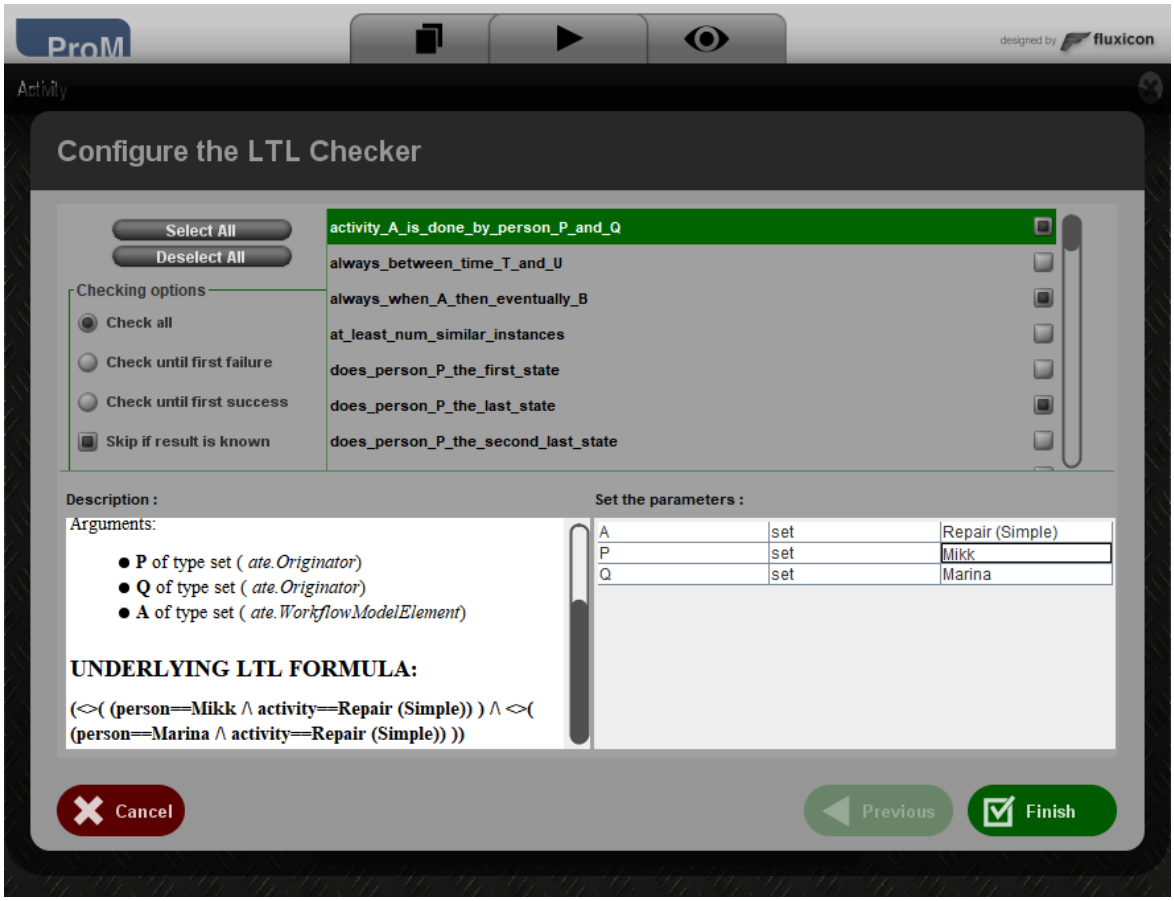
Joonis 5: Näidisfunktsioon pistikmooduli LTL Checker käsiraamatust [9]

Iga funktsiooni parameetri kohta kuvatakse ProM-i kasutajaliideses tekstiväli, kuhu kasutaja saab sisestada väärtuse, mis asendab LTL valemi töötlemisel vastava valemis sisalduva lokaalse muutuja. Lokaalsest muutujast kooloniga eraldatud atribuut määrab sisestatava väärtuse tüübi. Joonisel 6 on aluseks valem, millel on kolm parameetrit *A*, *P* ja *Q*, seega tekib

antud valemi kohta LTL Checkeri kasutusliideses ka kolm tekstivälja. Tekstivälja kõrval kuvatakse valemi kirjeldus ja valem ise, kus muutujad valemis asendatakse automaatselt tekstivälja sisestatud väärtustega. Funktsioon ei pea omama parameetreid, sellisel juhul ProM kasutajaliideses ühtegi väärtust valemis muuta ei saa.

Parameetritega funktsioone võib nimetada valemi mallideks, mis kirjeldavad valemi põhilise osa, samas jättes konkreetsed parameetrid kasutaja määrata. Sobivate mallide olemasolu korral peab kasutaja sisestama vaid mallis määratud parameetritele vastavad võrreldavad väärtused, mis omakorda lihtsustab LTL Checkeri kasutust märgatavalt. Näiteks kui üks kasutaja valmistab ette mingi protsessi spetsiifilised mallid, siis teised kasutajad saavad neid malle kasutada ka ilma LTL keelt valdamata.

Malli nimest ning kirjeldusest üldjuhul selgub, mida tekstiväljadesse sisestama peaks ning seeläbi ei pea kasutaja valemi loogikat täielikult mõistma. Pistikmooduliga LTL Checker töötamisel tuleb siiski kasuks valemite mõistmine ning oskus neid ise kirjutada, sest ilmselgelt ei kata mallid kõiki vajadusi. See-eest mallide kasutamine aitab mõista valemi matemaatilise keele olemust.

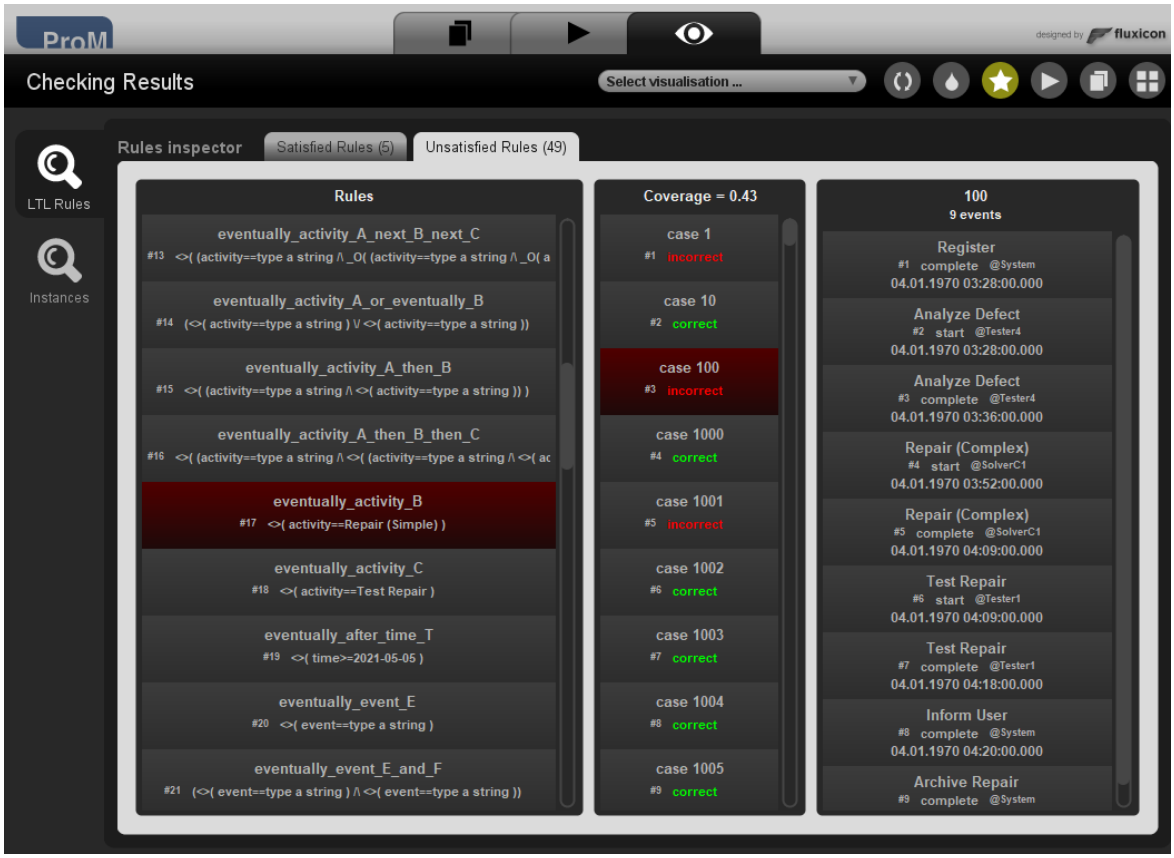


Joonis 6: LTL Checker Default mallide kasutamise vaade rakenduses ProM 6.10

Joonisel 6 on pistikmooduli LTL Checker Default esimene avanev vaade. Vaates on kasutada eeldefineeritud valemite mallid. Kontrollitakse ainult neid valemeid, mis on üleval paremal asuvas tabelis märgitud kasutades tabeli ridade lõpus olevaid märkeruute. Olulised on ka valemite kontrollimise valikud (*checking options*), mis võimaldavad täpsustata, kas kontrollida kõiki logijälgi või lõpetada kontrollimine pärast esimese mittekehtiva või kehtiva logijälje leidmist.

Peale vastavuskontrolli teostamist saab näha iga kontrollitud logijäljege kohta, kas valitud valem kehtib logijälje korral (*LTL Rules* vaade külgribal). Näide tulemuste vaatest on joonisel 7. Tulemuste vaates on valemid jaotataud kaheks. Valemid, mis kehtivad iga logijälje korral (*Satisfied Rules* sektsioon) ning valemid, mis on ei kehti vähemalt ühe logijälje korral (*Unsatisfied Rules* sektsioon). Valemile vajutades tekib keskmisesse veergu nimekiri logijägede nimedest ning märge, kas valitud valem kehtib vastavas logijäljes. Logijäle peale vajutades tekib parempoolsesse veergu sündmuse järjend valitud logijäljes.

Samuti saab näha logijälje kohta valemete kehtivust (*Inspector* vaade külgribal). *Inspector* vaade erineb *LTL Rules* vaatest selle poolest, et vasakpoolne ja keskmine tulp on omavahel vahetatud. See võimaldab iga logijälje kohta vaadata millised LTL reeglid on selles logijäljes rahuldatud ja milliste vastu on logijäljes eksitud.



Joonis 7: LTL Checkeri valemite kontrollimise tulemus rakenduses ProM 6.10

Järgnevalt vaatleme näitena peatüki „Vastavuskontroll“ lõpus kirjeldatud protsessi, mis sisaldab kahte sellist tegevust, mille sooritajaks ei tohi olla sama isik. Sellise protsessi reegli kehtivust saab kontrollida valemiga 3 (esitatud LTL Checker pistikmooduli sisendiks sobivas formaadis).

(3)
```
formula four_eyes_principle(a1:activity, a2:activity):={}
        forall[ p:person |(
             !(<> ( ( activity == a1 /\ person == p ))) \/
             !(<> ( ( activity == a2 /\ person == p )))
        )];
```

Siin tekib valemi kohta kaks tekstivälja, kus saab määrata millised on kaks tegevust *a1* ja *a2*. Järgnevalt kontrollitakse valemi kehtivust iga atribuudi *person* väärtusega. Kui isik sooritab mõlemat tegevust, saab valem tõeväärtuse väär. Kui isik sooritab ühte kahest tegevusest või mitte kumbagi, saab valem tõese tõeväärtuse. Kuna valem peab kehtima iga isiku korral, siis valemi kehtimiseks ei saa ükski isik sooritada ise mõlemat (parameetritega *a1* ja *a2* määratud) tegevust sama logijälje jooksul.

Käesolevas lõputöös integreeritakse pistikmooduli LTL Checker fuktsionaalsus rakendusse RuM. Lisatav funktsionaalsus võimaldab teostada LTL-põhist vastavuskontrolli samamoodi nagu see on võimalik rakenduses ProM. Lisaks funktsionaalsuse integreerimisele on käesoleva lõputöö raames loodud rakenduse RuM osana ka uus LTL Checker pistikmooduli kasutusliides, mis järgib RuM rakenduses hetkel kasutatavat stiili.

2. Kasutajaliidese disain

See peatükk annab ülevaate LTL Checkeri integreerimiseks loodud kasutajaliidese disaini protsessist. Täpsemalt on kirjeldatud disaini prototüüpimiseks kasutatud kujunduplatvorm Adobe XD, disaini eesmärgid, disaini iteratsioonid ja lõpplik kujundus koos ootuspärase funktionaalsusega.

2.1. Adobe XD kujundusplatvorm

Adobe XD [18] on rakendus, mis võimaldab kujundada ning luua interaktiivseid prototüüpe. Interakiivne prototüüp on loodava rakenduse töötamise imitatsioon ehk rakendusest on loodud mitmeid visandeid ning kasutajal on võimalik visandite vahel navigeerida selliselt, et navigeerimine imiteerib plaanitava kasutajaliidese käitumist. Rakendusse Adobe XD on integreerinud ka rakenduste Adobe Photoshop ja Illustrator funktsionaalsus, mis võimaldab nendes rakenduses muuta rakenduses Adobe XD loodud komponente ja ka vastupidi.

Interaktiivne prototüüp annab palju parema ettekujutuse rakenduse töötamisest kui lihtsalt erinevate vaadete pildid. See säästab hulganisti aega funktsionaalsuse seletamise pealt ning väldib olukorda, kus seletus on mitmeti mõistetav. Seega on vähem tõenäoline, et rakendust reaalselt implementeerides tuleb juurde ootamatuid muudatusi, mida on siis juba ajamahukam lisada.

Interaktiivsete prototüüpide loomiseks on mitmeid välimuse ja funktsionaalsuse poolest sarnaseid rakendusi, nagu Figma ja WebFlow. Kuid käesoleva prototüübi tegemiseks valiti nende seast mitme funktsionaalsuse olemasolu pärast siiski Adobe XD. Esimene selline funktsionaalsus on *Repeat Grid*. See võimaldab luua hulga korduvaid elemente valitud elemendist, kus ühe elemendi muudatused kajastuvad ka ülejäänutes. Lisaks sellele saab sellise elementide hulga tekstivälju muuta sinna peale tekstifaili lohistades. Seejärel saavad järjestikused elemendid väärtuse vastavalt tekstifaili sisule. Samamoodi saab sellises hulgas asendada ka pilte, sinna peale pildifaile lohistades. Teine selline funktsionaalsus on visandi ülemineku tüüp *Auto-Animate*. See teeb ülemineku ühelt visandilt järgmisele sujuvaks, võimaldades võrrelda kahe visandi elementide paigutust ning animeerides nende ümberpaiknemise. *Auto-Animate* funktsionaalsust on võimalik kasutada ka kasutajaliideses esinevate animatsioonide imiteerimiseks. Näiteks kasutab RuM paneele, millede nähtavale ilmumine toimub sujuva animatsioonina.

2.2. Disaini eesmärgid

Loodud kasutajaliidese kujunduses on lähtutud olemasoleva rakenduse kujundusest, sest tegemist on olemasoleva rakenduse uue sektsiooniga. Uus sektsioon on osa tervikust ning mõeldud ka kujunduselt tervikusse sobituma. Olemasoleva rakenduse kujundus on valdkonnale vastavalt asjalik ja karge ning hoiab läbivat minimalistlikku stiili, mida on jälgitud ka uue sektsiooni kujunduses.

Uue sektsiooni kujunduses on olemasolevast üle võetud struktuur, tekstide suurus, ikoonid ja värvid. Peamine eesmärk on võimaldada kasutada kogu vasavuskontrolli funktsionaalsust, mida rakenduse ProM pistikmoodul LTL Checker sisaldab. Ning võimalusel juurde lisada ka funktsionaalsust, mis teeks selle kasutamise mugavamaks.

Rakenduses RuM on paigas põhiliste elementide paigutus, mis peab kindlasti väljenduma ka uues sektsioonis. Iga sektsiooni juurde kuulub esiteks vasakus ääres olev sinine külgriba, kust saab navigeerida erinevatesse sektsioonidesse. Külgriba on alati nähtaval ning seda minimaliseerides kaovad sektsioonide nimetused ning nähtavaks jäävad sektsioonidele vastavad ikoonid, millede kaudu on võimalik sektsioonide vahel endiselt navigeerida.

Iga sektsiooni päises on vastava sektsiooni nimetus ning sellest paremal nupp, mis laseb kasutajal valida faili, mida vastava sektsiooni kasutamiseks vaja läheb. Faili valides tekib sektsiooni sisse vaheleht avatud failiga töötamiseks. Vahelehe vasakus ääres on valge taustaga külgriba, mis sisaldab valikuid kasutatava funktsionaalsuse parameetrite määramiseks. Parameetrite all on nupp, mis kontrollib parameetrite korrektsust ning käivitab seejärel sektsiooni funktsionaalsuse . Ülejäänud ala vahelehest kasutatakse tulemuse kuvamiseks ning tulemuste kuva ülesehitus võib erinevate sektsioonide vahel palju varieeruda. Antud töö lähtub põhiliselt vastavuskontrolli sektsiooni tulemuste kuva ülesehitusest, kuna see on lisatava LTL Checker funktsionaalsusega kõige sarnasem.

Rakenduse avalehel on küll navigeerimise külgriba, aga muid põhistruktuuri tunnuseid ei ole. Rakenduse avalehel on ka iga sektsioon eraldi välja toodud kasutades sektsioonile sobivat ikooni. LTL Checkeri jaoks uue sektsiooni loomisel on jälgitud sama stiili.

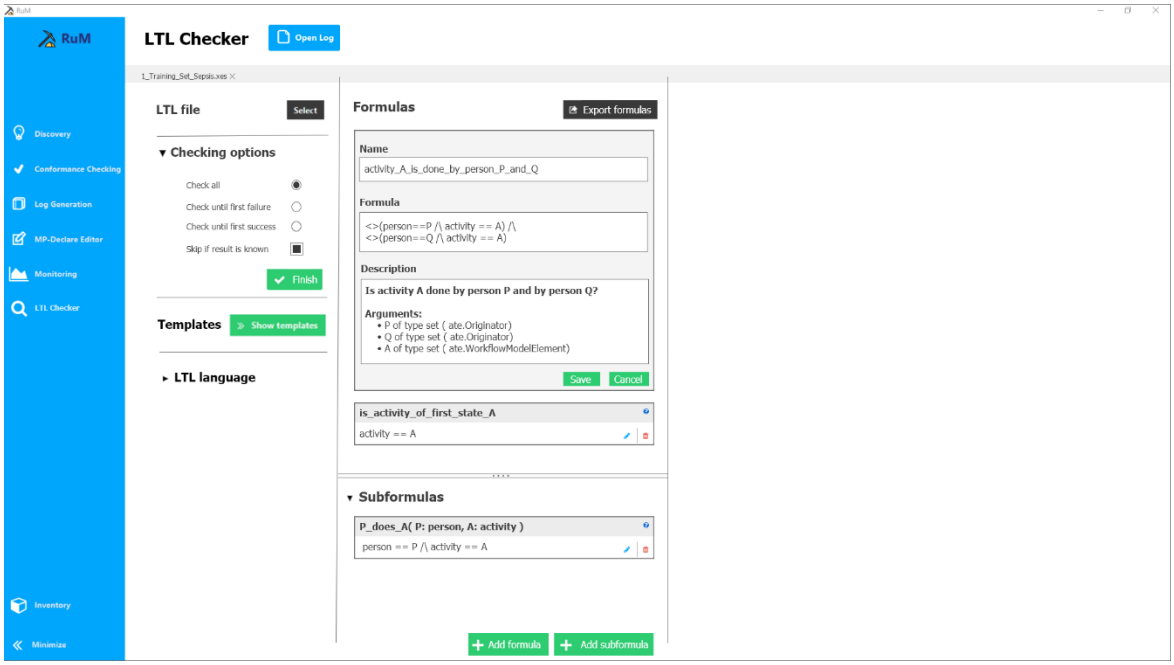
2.3. Kasutajaliidese iteratsioonid

Prototüübi esimene versioon on harva ideaalne, seega paratamatult peab läbima mitmeid iteratsioone sobiva lõpplahenduseni jõudmiseks. Üks iteratsioon kujutab endast protsessi,

kus kujundatakse kasutajaliidese vaated, võimalusel tehakse need interaktiivseks ning viiakse läbi prototüübi ülevaatus. Kui prototüübiga ollakse rahul, saab alustada reaalset implementeerimist Vastasel juhul jätkub arutelu osalejate vahel, selguvad puudujäägid ning täpsustatakse vajadusi. Loodav kujundus ei pruugi iga iteratsiooniga paremaks minna, kuid isegi vähem õnnestunud prototüüp on kasulik, sest see võimaldab kõrvutada varasemaid vaateid olemasolevaga, võimaldades paremini märgata kohti, mis teevad varasema versiooni paremaks. Töös läbiti sarnane prototüübi loomise protsess, mille käigus kohtuti RuM arendajatega korduvalt ja arutati vajaliku funktsionaalsuse sisu ning disaini muudatusi erinevate iteratsioonide vahel. Iteratsioonid väljanägemise poolest liiga palju ei muutunud ning rohkem oli vaja iteratsioonides tegeleda elementide paigutusega.

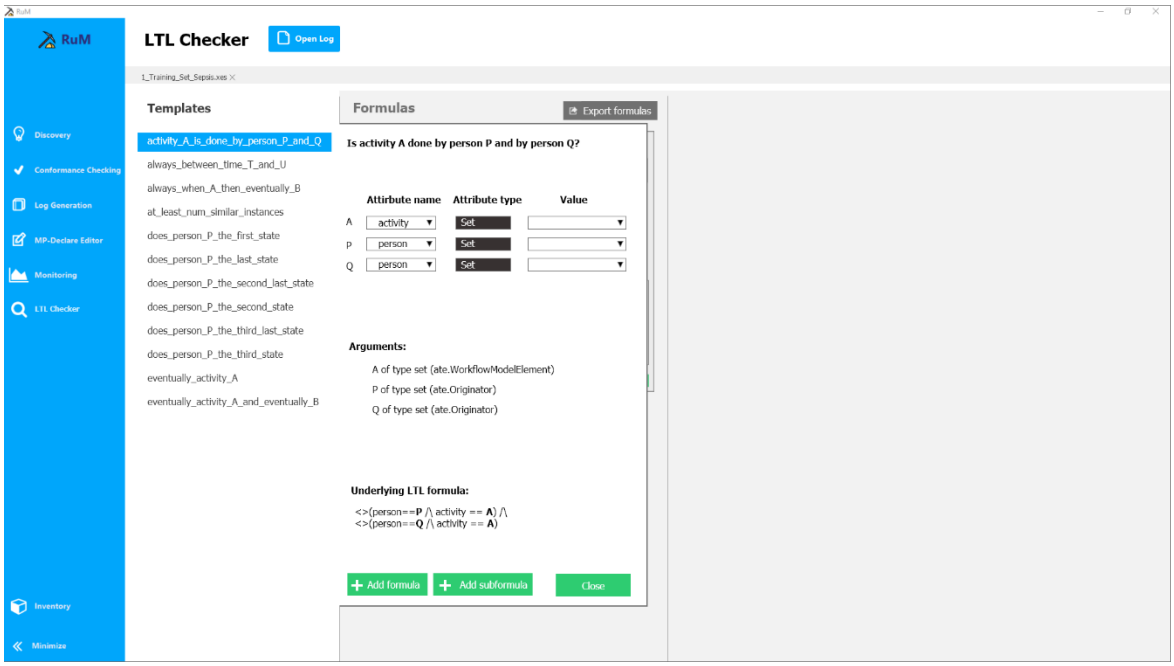
Uue sektsiooni sisuks on erinevad LTL keele valemid, nende redigeerimine, valemite põhjal loodud mallide parameetrite määramine ja vastavuskontrolli tulemuste kuvamine. Valemeid on mõistlik esitada nimekirjas ning hõlbustamaks valemeid üksteisest eristada on kujundusse lisatud helehall taustapind. Helehallil taustal on kirjas valemi nimetus. Kuna helehall taust vähendab musta kirja kontrastsust, on valemi nimi helehallil taustal paksus kirjas. Samuti on eristatud valemi seisund, kus valemi sisu muudetakse. Seal aitab helehall taust hoomata valemi erinevaid osasid.

Nagu joonisel 8 on näha, saab lisaks valemitele lisada ka abivalemeid (*subformula*). Abivalemite kasutamine aga ei jäänud lõplikusse versiooni, sest nende kasutamine tundus pigem segadusttekitav. Samuti jäi välja külgribas avatav menüü *LTL language*, mis oleks sisaldanud LTL keele kirjeldust.



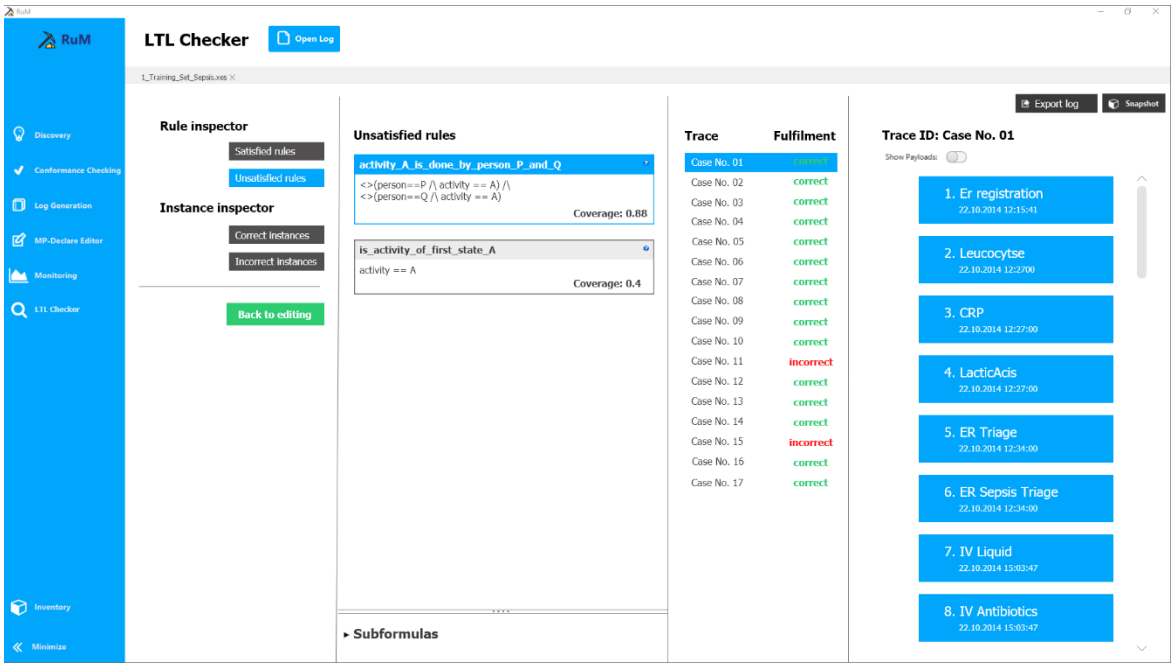
Joonis 8: Valemite muutmise vaade

Lisaks on loodud eraldi vaade valemite mallide kasutamiseks. Näide vaatest on joonisel 9. Mallide vaates asendub külgriba nimekirjaga mallidest, kus sinine taust tähistab valitud valemit. Paremal on info malli kohta. Selgelt on eristatud valemi nimetus, sisestatavad atribuudid, malli kirjeldus ning kasutatav valem. Kolmnurk atribuudi nime ja väärtuse veeru lahtris tähistab rippmenüüd. Valikutes on sündmuste logis olemasolevad atribuudid ning valitud atribuudi vastavad väärtused.



Joonis 9: Valemi lisamise vaade

Tulemuste vaade on funktsionaalsuse ja elementide paigutuse poolest peaaegu identne pistikmooduli LTL Checker kasutajaliidesega rakenduses ProM. Põhiline muutus on siin vaid elementide välimus ning seegi liiga palju ei erine. Valitud elemendi tähistus on punase tausta asemel sinine taust, mis on rakendusele RuM omane. Logijälgede tulbas on vastavuskontrolli tulemus nimetuses paremal, mitte nimetuse all. Sündmuste veerg on üks ühele üle võetud sektsioonist Conformance Checking. Erinevalt pistikmooduli LTL Checker kasutajaliidesest saab siin näha ka muid sündmuse juurde kuuluvaid atribuute ja nende väärtuseid, selleks on *Show Payloads* nupp. Tulemuste vaade on joonisel 10.



Joonis 10: Tulemuste vaade

See iteratsioon on vastuolus rakenduse RuM ühe põhistruktuuri tunnusega. Vaadetes on külgriba küll olemas ning sisaldab parameetreid, aga selle sisu ei tohiks erinevates vaadetes muutuda. Kuna külgribale peab jääma vaid oluline, tuli ülejäänud elemendid mujale paigutada.

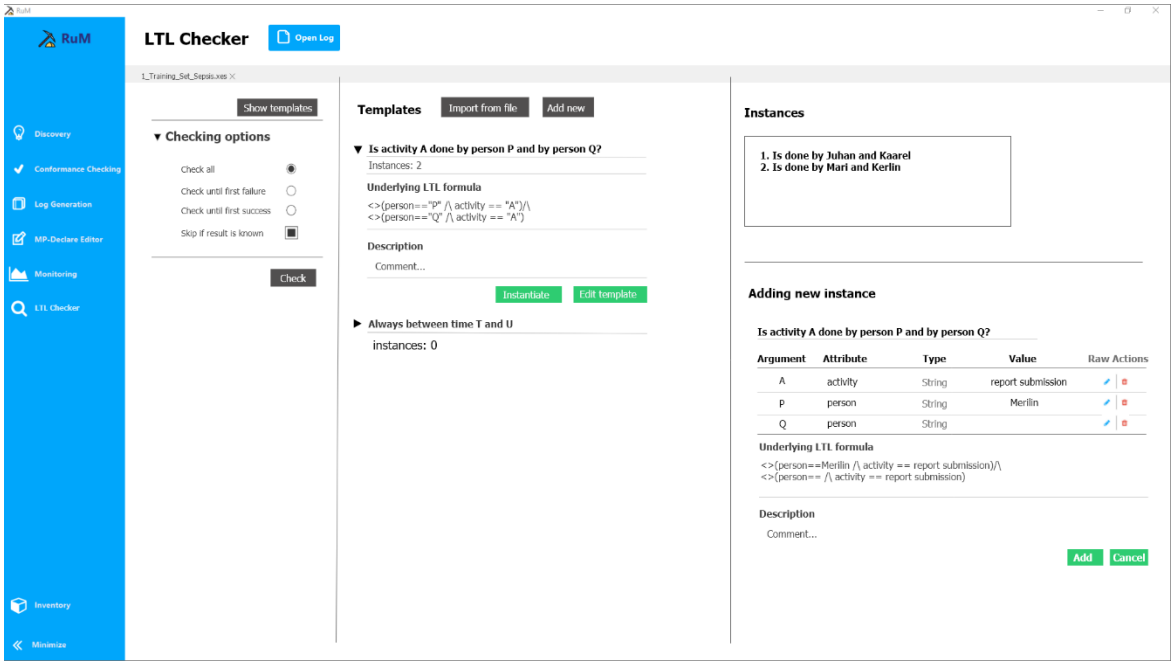
2.4. Lõplik kujundus

Kuna funktsionaalsus on suunatud vasavuskontrolli teostamiseks, siis kasutajaliidese kujundamisel on aluseks võetud rakenduse RuM olemasolev *Conformance Checking* sektsioon. Seeläbi on olemasolevat vastavuskontrolli sektsiooni kasutanud kasutajale uus sektsioon juba tuttava struktuuriga ning uue sektsiooni kasutama õppimine lihtsam. Samuti sarnaneb kasutajaliides kohati pistikmooduli LTL Checker kasutajaliidesega rakenduses ProM, mida selles peatükis edaspidi nimetatakse ProM versiooniks.

RuM rakenduses lisatud uus funktsionaalsus on koondatud uude sektsiooni *LTL Checker*. Selle sektsiooni kasutamiseks on esiteks vaja sisendina anda sündmuslogi ning seejärel on vaja määrata vastavuskontrolliks kasutatavad LTL keele eeskirjad. Peale sündmuslogi valimist avaneval vahelehel on olemas külgriba ning veerg *Templates*. Külgriba sisaldab vastavuskontrolli üldiseid parameeterid ning veerg *Templates* sisaldab vahetult peale sündmuslogi avamist kahte nuppu: *Import from file* ja *Add new*. Esimene nupp avab failisüsteemi ning võimaldab valida LTL valemite malle sisaldava faili (laiendga .ltl). Peale faili valimist kuvatakse nimekiri failis sisalduvatest valemi mallidest ning kasutaja saab valida, milliseid malle vastavuskontrollis kasutada. Valitud mallid lisanduvad *LTL Checker* vahelehel veergu *Templates*. On võimaldatud ka mitmest erinevast failist mallide lisamine ning vajadusel saab veergu nupuga *Add new* lisada tühja malli. Tühi mall annab võimaluse valemit redigeerida.

Kontrollitavate valemite lisamiseks tuleb esiteks vajutada mallile, mida soovitakse valemi loomisel kasutada. Malli peale vajutamine avab malli täpsema info, mis sisaldab ülevaadet juba loodud valemitest. Uue valemi lisamiseks tuleb vajutada nuppu *Instantiate*. Nupu vajutamise järel tekib paremale aken *Adding new instance*. Siin saab määratleda malli parameetrid ning lisada loodud valem vajutades nuppu *Add*. Seejärel tekib lisatud valemi nimetusega kirje üleval paiknevasse *Instances* nimekirja. Valemi malli saab kasutada korduvalt valemite defineerimiseks ning kõik valemid, mis on valitud malliga defineeritud kajastuvad *Instances* nimekirjas. See on ProM versiooniga võrreldes uus funktsionaalsus, sest ProM versioonis sai iga malli kasutada vaid ühe valemi defineerimiseks.

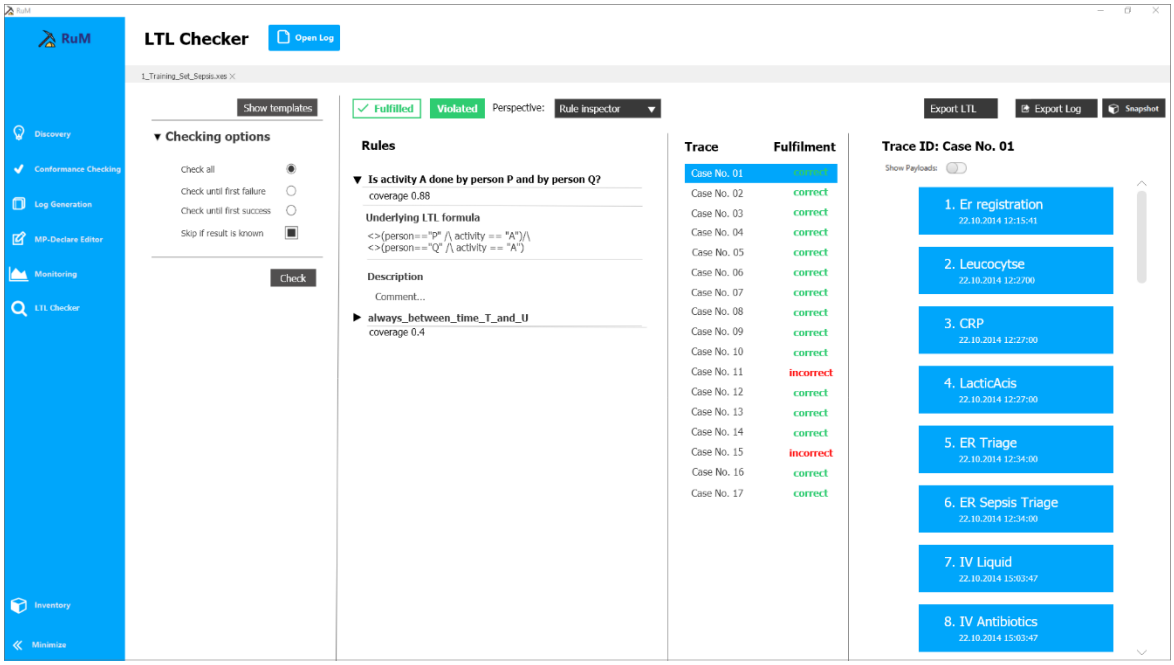
Joonisel 11 on näide sektsioonist *LTL Checker*. Lisatud on kaks malli *Is activity A done by person P and by person Q?* ja *Always between time T and U*. Esimese malli põhjal on pooleli uue valemi määratlemine ning sissekirjutatud väärtused kajastuvad allpool paiknevas valemis, asendades vastavad muutujad. Väärtuste sisestamise tabeli kujundus jälgib sama stiili nagu sektsioonis *MP-Declare Editor* ksutatavad tabelid. *Instances* nimekirjas on näha, et selle malliga on juba defineeritud kaks valemit nimetustega *Is done by Juhan and Kaarel* ja *Is done by Mari and Kerlin*. See arv kajastub ka *Templates* veerus



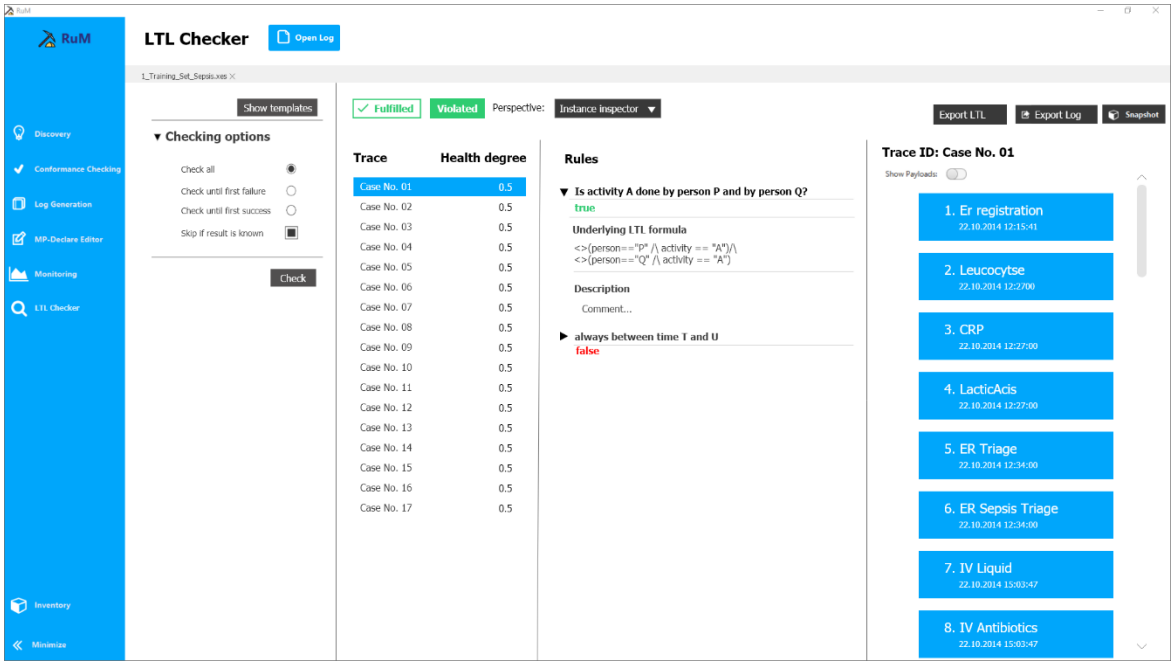
Joonis 11: Valemi mallide kasutamise vaade

Olles defineerinud valemid saab neid kontrollida vajutades nuppu *Check* külgribal. Külgribast paremal olev ala asendub seeläbi tulemuste vaatega. Võrreldes eelnevat tulemust vaate iteratsiooniga (Joonis 10) on jäänud samaks logijälgede ja sündmuste veerud, aga lõplikus lahenduses (Joonis 12) on vasakult teine veerg *Rules* ümber kujundatud.

Samamoodi nagu ProM versioonis, on RuM uue sektsiooni lõplikus kujunduses olemas *Rule inspector* (Joonis 12) ja *Instance inspector* (Joonis 13) vaated. Tulemuste vaates saab nende vahel navigeerida kasutades üleval paiknevat rippmenüüd. Kehtivate ja mittekehtivate valemite vaatamiseks on üleval kaks rohelist nuppu *Fulfilled* ja *Violated*, mis käituvad kui tulemuste filtrid.



Joonis 12: Tulemuste *Rule inspector* vaade



Joonis 13: Tulemute *Instance inspector* vaade

Tulemuste vaates on ka mitu lisafunktsionaalsust, mis ProM versioonis puuduvad. Üks selline funktsionaalsus on võimalus eksportida vastavuskontrolli tulemust sündmuste logi fromaadis XES kasutades nuppu *Export Log*. Lisaks on kasutajaliideses ka nupp *Snapshot*, mis salvestab vastavuskontrolli tulemuse vahetulemusena otse RuM rakendusse, mida saab teistes sektsioonides ilma vastavat faili salvestamata ja uuesti importimata kasutada. Sama-sugune logi eksportimise võimalus on olemas mitmes rakenduse RuM sektsioonis.

Sekstiooni *LTL Checker* omane funktsionaalsus on võimalus eksportida vastavuskontrolli sisendina kasutatavaid eeskirju .ltl laiendiga formaadis. Eksportida saab nii parameetritega malle, kui ka määratletud väärtustega valemeid. See võimaldab kasutatud valemeid salvestada ning hiljem taaskasutada. Lisafunktsionaalsus on ka pärast vastavuskontrolli liikuda tagasi valemite muutmise vaatesse, selleks on nupp *Show templates*. Kui ProM versioonis tehakse valemi määratlemisel kirjaviga ning seetõttu saadakse vigane tulemus, siis ainus võimalus selle parandamiseks on kogu valemite määratlemise protsessi alustada päris algusest. Tagasiminemise võimalus teeb sarnase olukorra lahendamise rakenduse RuM versioonis palju kasutajasõbralikumaks.

3. Implementatsioon

See peatükk kirjeldatab rakenduse RuM arhitektuuri ja LTL põhise vastavuskontrolli sektsiooni implementatsiooni.

RuM lähtekoodi muudatused on kättesaadavad aadressil: https://bitbucket.org/doorless1634/thesis/commits/tag/v1.0.0. Vastav kompileeritud RuM rakenduse version koos LTL vastavuskontrolli sisendfailide näidistega on kättesaadav aadressil: https://drive.google.com/drive/folders/1EeC2o425WfUr5Ahc6KyXGuPzHCkDX5X5?usp=sharing.

3.1. Arhitektuur

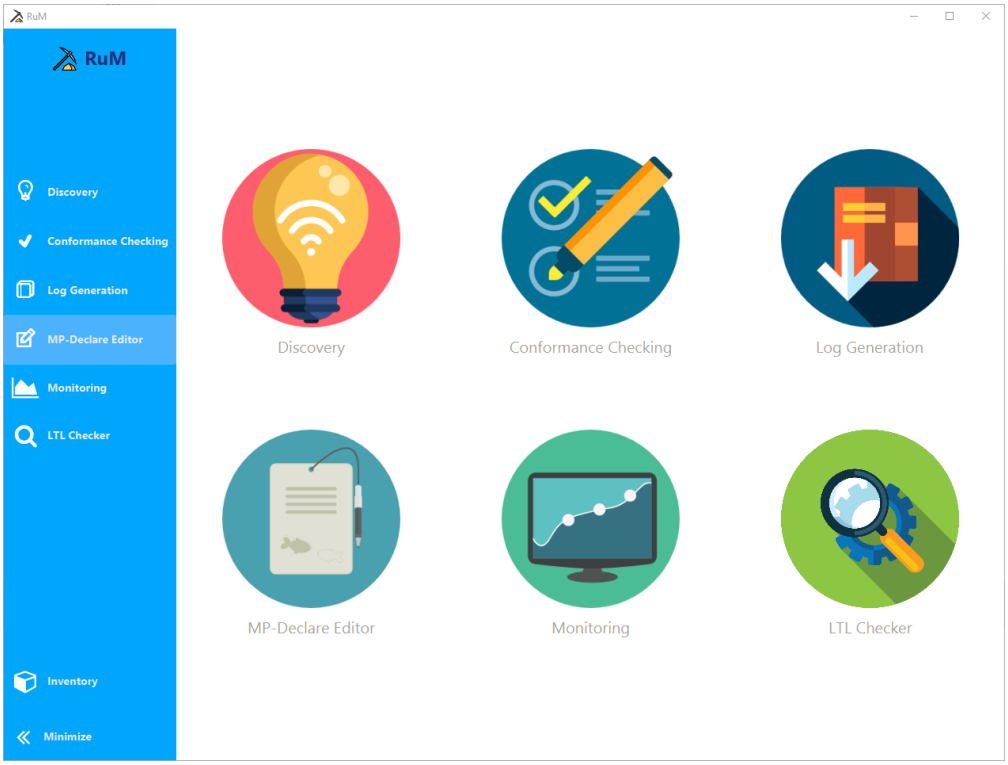
RuM on Java-põhine rakenus, mis on arendatud kasutades Java versiooni 11. Rakenduse arhitektuur põhineb mudeli-vaate-kontrolleri (Model-View-Controller) disanimustril [19]. Vaade on visuaalne pool rakendusest ehk kõik, mis on kasutajale nähtaval ja millega kaudu kasutaja ülejäänud rakenudsega suhtleb. Mudeli osa sisaldab kogu rakenduse äriloogikat. Antud rakenduse puhul hõlmab see endas protsessikaeve algoritme, andmete töötlust ning kasutatavaid teeke. Kontroller on vahekomponent äriloogika ja kasutajaliidese vahel. Kontroller vastutab kasutajaliidese muutumise eest ehk reageerib kasutaja interaktsioonile.

Kasutajaliides on loodud kasutades JavaFX-raamistikku. JavaFX kasutab vaadete ülesehituse salvestamiseks formaati FXML. Iga koodibaasis sisalduv FXML fail (laiendiga .fxml) vastab konkreetsele sektsiooni vaatele või mõnele väiksemale vaadetes kasutatavale komponendile. Kasutajaliide loomisel kasutati rakendust Scene Builder (versiooni 16.0.0), mis võimaldab visuaalselt kujundada rakenduse kasutajaliidest. Scene Builder võimaldab kasutajaliidese komponentide liimist, kujundamist ja paigutamist ilma FXML koodi kirjutamata. Rakenduse elementide välimus (värvid, teksti suurus, jne) on kirjeldatud kasutades rakenduseüleseid CSS stiililehti, mis teeb ühtlase rakenduseülese kujunduse jälgimise oluliselt lihtsamaks.

Protsessikaeve Algoritmid ja muu vajalik on rakendusele lisatud Maveni sõltuvusena (*dependency*). Maven Repository ei sisalda kõiki vajalikke sõltuvusi, seega on koodibaasi loodud lokaalne Maveni sõltuvuste hoidla (*repository*). Sõltuvused on sinna lisatud kompileeritud .jar failide kujul. Rakenduse ProM pistikmoodul LTL Checker lisati ka lokaalse sõltuvusena koodibaasi.

3.2. Funktsionaalsus

Käesoleva töö raames valimis algne versioon uuest LTL-põhise vastavuskontrolli sektsioonist. Valminud sektsiooni tulemuste vaate funktsionaalsuset on palju olemas, aga valemi mallide kasutamise vaade on veel suhteliselt algelises arengujärgus. Põhilised struktuuri elemendid on olemas ning töötavad. Avalehel on uuele sektsioonile vastav ikoon ning navigatsiooni külgribal lisandunus sektsioon *LTL Checker* (joonis 14).



Joonis 14: RuM avaleht

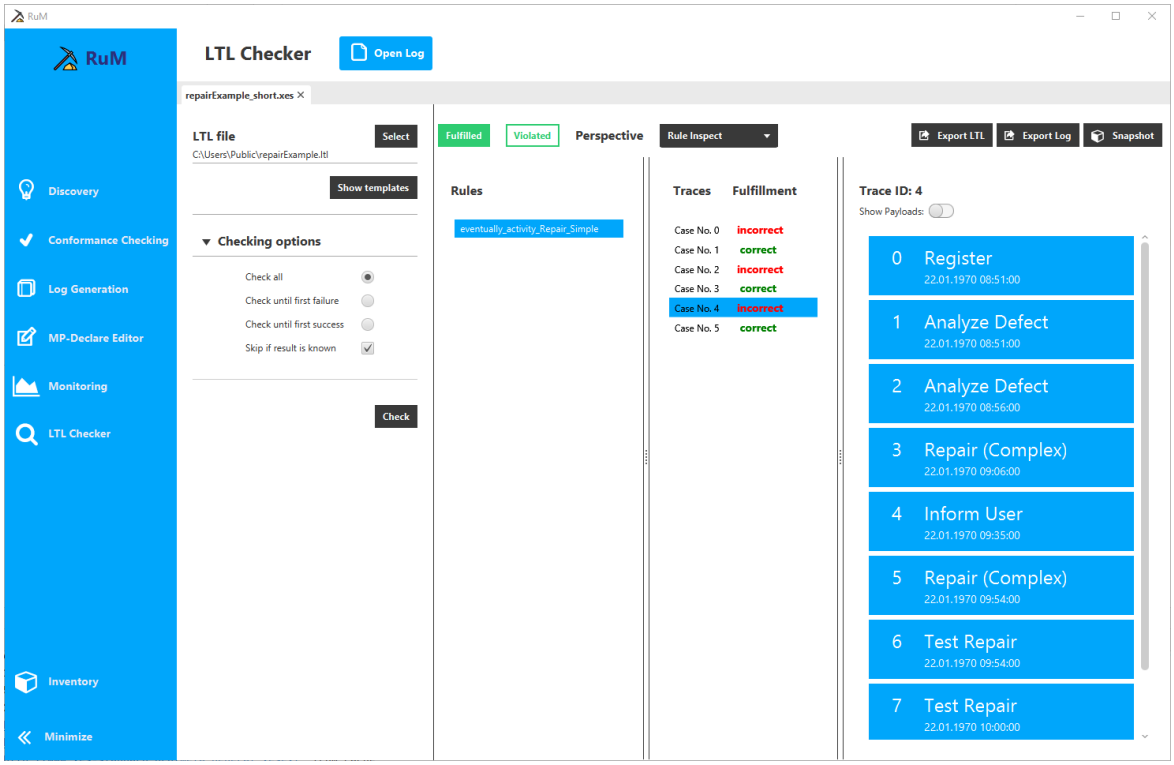
Sektsiooni *LTL Checker* päises on vastava sektsiooni nimetus ja sündmuslogi avamise nupp. Peale sündmuslogi valimist tekib uus vaheleht avatud failiga töötamiseks. Sarnaselt rakenduse teiste sektsioonidega on ka uues vastavuskontrolli sektsioonis võimalik samaaegselt avada mitu sündmuslogi faili erinevatel vahelehtedel ning nende vahelehtede vahel vabalt navigeerida.

Erinevalt lõplikus kujunduses ettenähtust on tulemuste vaate külgribal hetkel nupp *Select*, mis võimaldab valida LTL valemi faili. Siin eeldatakse, et sisendina saadavad valemid on juba määratletud. Vastavuskontrolli jaoks kasutatakse kõiki valemeid failis. Selline LTL faili valimise võimalus on ajutiselt tulemuse vaates, see imiteerib sisendi saamist hetkel

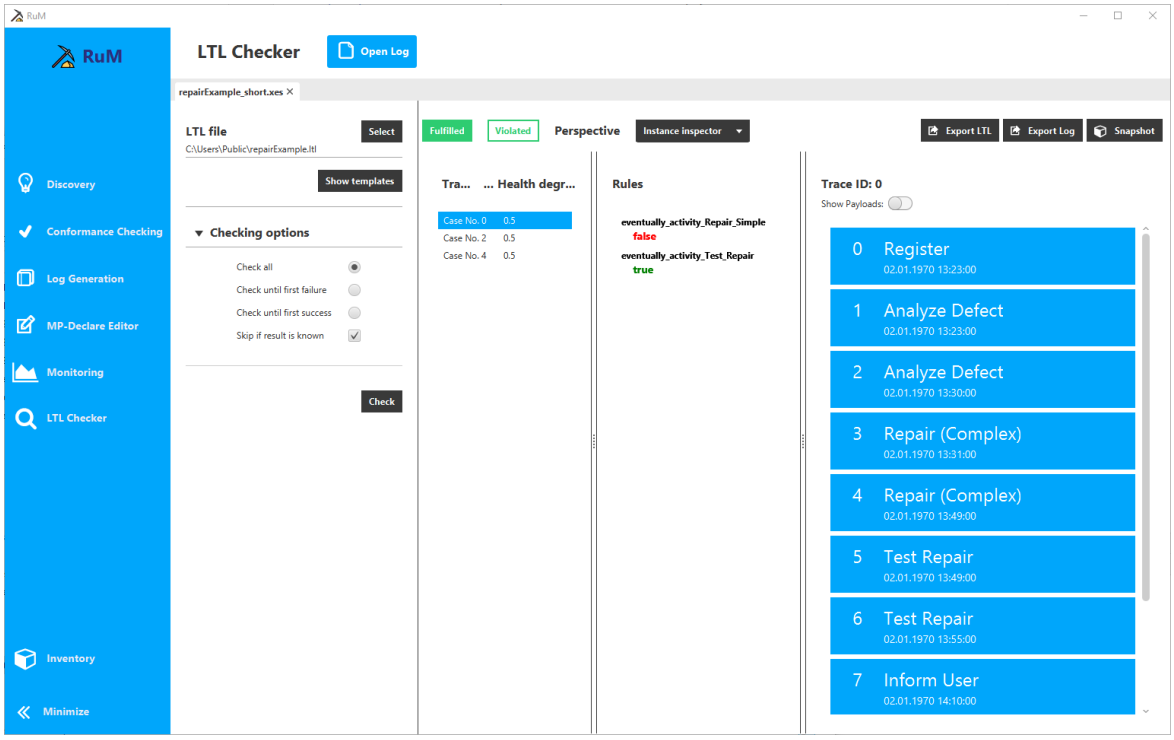
puuduolevast valemi mallide kasutamise vaatest. Peale LTL valemite faili valimist käivitab nupp *Check* vastavuskontrolli.

Pistikmooduli LTL Checker funktsionaalsuse RuM tulemuste vaatesse integreerimine on valmis. Töötab valemite kasutamine sündmuslogi vastavuskontrolliks, külgribal *Checking options* sees olevad valikud mõjutavad tulemust ning tulemuse kuvamine veergudes toimub samamoodi nagu lõplikus prototüübis ette nähtud. RuM rakendusse integreeritud LTL Checkeri poolt tagastatavaid tulemusi on võrreldud ProM pistikmooduli poolt tagastatud tulemustega ja käesoleva töö kirjutamise hetkel ei ole tulemustes ühtegi erisust tuvastatud.

Lõplikus kujunduses ettenähtuga võrreldes on ainsaks erisuseks veeru *Rules* kujundus ja käitumine. Prototüübis on ette nähtud, et veerus oleva valemite nimetusele vajutades kuvatakse selle all info valemi kohta. Valminud versioonis on veerus ainult reeglite nimetused. Südnmuste nimekirjas töötab ka nupp *Show Payloads*. Tulemuste vaadete *Rule Inspector* (joonis 15) ja *Instance Inspector* (joonis 16) vahel saab musta rippmenüüga liikuda.



Joonis 15: Implementeeritud tulemuste *Rule inspector* vaade



Joonis 16: Implementeeritud tulemuste *Instance inspector* vaade

Mõlemas vaates saab tulemusi filtreerida kasutades üleval paiknevaid rohelisi nuppe *Fulfilled* ja *Violated*. Nupud võimaldavad kasutajal (sarnaselt ProM pistikmooduli kasutusliidesele) keskenduda nendele logijälgedele mis vastavad sisendiks olnud valemitele täielikult või nendele logijälgedele mis rikuvad vähemalt ühte valemit.

Vaate üleval paremal nurgas paiknevad nupud *Export LTL*, *Export Log*, *Snapshot*. Nendest esimene on mõeldud LTL valemite eksportimiseks. Ning järgnevad kas on mõeldud filtritele vastavatest logijälgedest koosneva sündmuslogi eksportimiseks (nupp *Export Log*) ja vahetulemusena otse RuM rakendusse salvestamiseks (nupp *Snapshot*).

4. Kokkuvõte

Bakalaureuse töö eesmärk oli täiendada protsessikaeve rakenduse RuM funktsionaalsust ning disainida lisatavale funktsionaalsusele vastav kasutajaliidese osa. RuM on deklaratiivseid protsessikaeve metoodikaid kasutav raamistik. Kasutajate soov oli lisada rakendusse LTL keelel põhinev vastavuskontrolli funkstionaalsus. Olemasolevad vastavuskontrolli metoodikad põhinevad keelel Declare, kus on kasutada piiratud arv selge semantilise tähendusega eeskirju. LTL keel on (võrreldes Declare keelega) palju väljendusrikkam ning võimaldab kirjeldada oluliselt keerukamaid protsessi eeskirju.

Vastavuskontrolli põhimõte on protsessi eeldatava ja tegeliku käitumise võrdlemine ning sellest saadud info põhjal protsessi parendamine. Vastavuskontroll tugineb automatiseeritud algoritmidel, mis võrdlevad sündmuslogi sisu ja protsessi töökäigu mudelit. Soovitud funktsionaalsuse saavutamiseks integreeriti antud töös rakendusse RuM rakenduse ProM pistikmoodul LTL Checker. Tulemusena muutus pistikmooduli funktsionaalsuse kasutamine rohkem võimalusi pakkuvaks ning kasutajale mugavamaks.

Et olemasoleva rakenduse kujundusega sobituda, tuli lisatud pistikmooduli kujundust muuta. Kujunduse muutmisel alustati disaini prototüübi loomisest kasutades kujundusplatvormi Adobe XD. Disaini prototüübi loomisel võeti iteratiivne lähenemine, mis aitas tagada, et disaini põhiosad oleksid enne arenduse algust paigas ning kõigi osapooltega kokku lepitud. Lõplik kujundus sai sarnane rakenduses RuM olemasoleva vastavuskontrolli sektsiooni kujundusega. Kuna uus sektsioon on kasutajale juba tuttava struktuuriga, lihtsustab see uue sektsiooni kasutama õppimist. Tööprotsess toimus tihedas koostöös rakenduse RuM arendajatega. Praktilise kasutaja tagasiside andis parema ettekujutuse, mida tegelikult vajati.

Töös on kirjeldatud kasutajaliidese kujundamise protsessi ning tehtud algust reaalse implementatsiooniga. Käesoleva töö tulemusena olemasolevasse rakendusse lisatud pistikmooduli LTL Checker funktsionaalsus üldjoontes toimib ning moodustab vastavuskontrolli kontekstis loogilise terviku. Tulevikus on plaanis mõningate täiendavate funktsionaalsuste lisamine ning nendele funktsionaalsustele vastav disaini lahendus on käesoleva töö raames juba loodud.

Viidatud kirjandus

[1] Alman A, Ciccio C Di, Haas D, Maggi FM, Nolte A. Rule mining with RuM. Proceedings - 2020 2nd International Conference on Process Mining, ICPM 2020. 2020. https://www.researchgate.net/publication/346375422_Rule_Mining_with_RuM

[2] RuM – Rule Mining Made Simple. https://rulemining.org/ (04.05.2021)

[3] ProM Tools http://www.promtools.org/doku.php (04.05.2021)

[4] Van der Aalst W. Process Mining: Data Science in Action. 2nd ed. Springer Publishing Company, Incorporated; 2016.

[5] Van der Aalst W, Adriansyah A, de Medeiros AKA, Arcieri F, Baier T, Blickle T, et al. Process Mining Manifesto. In: Daniel F, Barkaoui K, Dustdar S, editors. Business Process Management Workshops Berlin, Heidelberg: Springer Berlin Heidelberg; 2012. p. 169–94. https://pure.tue.nl/ws/portalfiles/portal/3707913/671825010972295.pdf

[6] Kang YS, Park S. A Study of Process Mining-based Business Process Innovation. Procedia Comput Sci. 2016;91:734–43. https://www.researchgate.net/publication/305953976_A_Study_of_Process_Mining-based_Business_Process_Innovation

[7] Aalst W, Pesic M, Schonenberg H, Sidorova N. Constraint-Based Workflow Models: Change Made Easy. In 2007. p. 77–94. https://www.researchgate.net/publication/220830907_Constraint-Based_Workflow_Models_Change_Made_Easy

[8] Pesic M. Constraint-based workflow management systems : shifting control to users. In 2008. https://pure.tue.nl/ws/files/2815085/200811543.pdf

[9] De Beer HT. The LTL Checker Plugins a (reference) manual. 2005. http://www.processmining.org/_media/documentation/conformance/ltlchecker-manual.pdf

[10] Van Der Aalst WMP, De Beer HT, Van Dongen BF. Process Mining and Verification of Properties: An Approach based on Temporal Logic http://www.processmining.org/_media/publications/aalst2005f.pdf (04.05.2021)

[11] IEEE 1849-2016 XES Standard. https://xes-standard.org/ (04.25.2021)

[12] Dolech D, Eindhoven AZ, Günther CW, Verbeek E. XES Standard Definition 2014 https://pure.tue.nl/ws/files/3981980/692728941269079.pdf (04.25.2021)

[13] Burattin A, Maggi F, Sperduti A. Conformance Checking Based on Multi-Perspective Declarative Process Models. Expert Syst Appl. 2015;65. https://www.researchgate.net/publication/273788691_Conformance_Checking_Based_on_Multi-Perspective_Declarative_Process_Models

[14] de Leoni M, Maggi F, Aalst W. Aligning event logs and declarative process models for conformance checking. In 2012. p. 82–97. http://www.processmining.org/_media/publications/bpm2012.pdf

[15] Process. Mining. ProM. http://www.processmining.org/prom/start (04.10.2021)

[16] Aalst W, Dongen B, Verbeek H, Weijters A. The ProM Framework: A New Era in Process Mining Tool Support. In: Lecture Notes in Computer Science. 2005. p. 444– 54. https://www.researchgate.net/publication/220783318_The_ProM_Framework_A_New_Era_in_Process_Mining_Tool_Support

[17] prom - Revision 44717: /Packages/LTLChecker/Trunk https://svn.win.tue.nl/repos/prom/Packages/LTLChecker/Trunk/ (04.12.2021)

[18] Adobe XD. Fast & Powerful UI/UX Design & Collaboration Tool. (05.05.2021). https://www.adobe.com/products/xd.html

[19] Krasner G, Pope S. A Description of the Model-View-Controller User Interface Paradigm in the Smalltalk80 System. J Object-oriented Program - JOOP. 1988;1. https://www.researchgate.net/publication/239452280_A_Description_of_the_Model-View-Controller_User_Interface_Paradigm_in_the_Smalltalk80_System

Lisad

I. Litsents

Lihtlitsents lõputöö reprodutseerimiseks ja üldsusele kättesaadavaks tegemiseks

Mina, **Kaspar Kadalipp**,

(autori nimi)

1. annan Tartu Ülikoolile tasuta loa (lihtlitsentsi) minu loodud teose

LTL-põhise vastavuskontrolli integreerimine protsessikaeve rakendusse RuM,

(lõputöö pealkiri)

mille juhendajad on **Fabrizio Maria Maggi** ja **Anti Alman**,

(juhendaja nimi)

reprodutseerimiseks eesmärgiga seda säilitada, sealhulgas lisada digitaalarhiivi DSpace kuni autoriõiguse kehtivuse lõppemiseni.

2. Annan Tartu Ülikoolile loa teha punktis 1 nimetatud teos üldsusele kättesaadavaks Tartu Ülikooli veebikeskkonna, sealhulgas digitaalarhiivi DSpace kaudu Creative Commonsi litsentsiga CC BY NC ND 3.0, mis lubab autorile viidates teost reprodutseerida, levitada ja üldsusele suunata ning keelab luua tuletatud teost ja kasutada teost ärieesmärgil, kuni autoriõiguse kehtivuse lõppemiseni.

3. Olen teadlik, et punktides 1 ja 2 nimetatud õigused jäävad alles ka autorile.

4. Kinnitan, et lihtlitsentsi andmisega ei riku ma teiste isikute intellektuaalomandi ega isikuandmete kaitse õigusaktidest tulenevaid õigusi.

Kaspar Kadalipp

07.05.2021