TARTU ÜLIKOOL
Matemaatika-informaatikateaduskond
Arvutiteaduse instituut

Rivo Roo

Veebirakenduste mudelipõhine testimine asukohapõhise tarkvara näitel

Magistritöö (60 EAP)

Juhendajad: teadur Juhan Ernits

dotsent Helle Hein

Autor: ……..……. „ …. “ märts 2010

Juhendaja: …..…… „ …. “ märts 2010

…… „ …. “ märts 2010

Lubada kaitsmisele

Professor: ……..……..…..…. „ …. “ märts 2010

Tartu 2010

Sisukord

Sissejuhatus

Tarkvaraorganisatsioonid pööravad testimisele üha rohkem tähelepanu. Paljud tarkvaraettevõtted tunnistavad, et puudulik testimine on neile põhjustanud majanduslikku kahju, kuna mida hiljem viga avastatakse, seda kallim on selle parandamine [1]. Testimise osatähtsuse suurenedes on tarkvarafirmades suundumus üha rohkem kasutada automaattestimist.

Üheks viimastel aastatel populaarsust kogunud automaattestimise meetodiks on mudelipõhine testimine. Lähenemist nimetatakse mudelipõhiseks, kuna testitavale süsteemile esitatud nõuete alusel luuakse mudel ning viimasest tuletatakse testid. Testimise käigus kontrollitakse mudeli ja testitava süsteemi omavahelist vastavust. Mudelipõhist lähenemist rakendatakse eelkõige funktsionaalsetes testides. Antud meetodi eeliseks võrreldes traditsiooniliste automaattestimise meetoditega on lihtsamini saavutatav testide katvus, kuna testlugusid ei ole vaja ükshaaval käsitsi kodeerida, vaid neid saab suurel hulgal genereerida mudeli alusel.

Mudelipõhist testimist kasutatakse mitmetes suurtes ja ülemaailmselt tuntud ettevõtetes, mille hulka kuuluvad Microsoft, IBM, Google, BMW jt. Ometi ei ole mudelipõhine lähenemine vaatamata oma teoreetilistele eelistele veel praktikas kuigi laialt levinud [2], [3]. Samuti on Eestis mudelipõhist testimist seni vähe rakendatud.

Antud magistritöö koostamise käigus katsetati mudelipõhist testimist ühe konkreetse kommertstoote – asukohapõhise tarkvara WFM (Reach-U WorkForce Management) testimisel. WFM on veebirakendus, mis võimaldab logistikaettevõtetel oma töötajate liikumist planeerida, kasutades töötajate asukohtade määramiseks mobiilpositsioneerimist. Rakendus töötab integreerituna mobiilioperaatori infrastruktuuri ning suhtleb mitme välise süsteemiga, näiteks mobiilioperaatori arveldussüsteemi ja SMS-keskusega.

Süsteemi WFM oli varasemalt testitud muude meetoditega ning tekkis vajadus mõnd mudelipõhise testimise vahendit kasutades välja selgitada, kuidas aitab mudelipõhine lähenemine rakenduse kvaliteeti parandada ning kui suur ressurss sellele kulub.

Magistritöö eesmärk oli hinnata mudelipõhise lähenemise sobivust süsteemi WFM ning analoogiliste veebirakenduste testimisel. Kriteeriumiks oli varasema testimisega avastamata vigade leidmine, mille tulemusel tõsteti süsteemi üldist kvaliteeti.

Magistritöö on jagatud viieks peatükiks.

Töö esimeses peatükis antakse ülevaade mudelipõhise testimise teooriast. Peatükk koostati teemakohase kirjanduse alusel.

Töö teises peatükis vaadeldakse mudelipõhist testimist võimaldavaid vahendeid ning tehakse nende hulgast valik antud töös kasutamiseks.

Töö kolmandas peatükis antakse ülevaade testitavast süsteemist WFM. Töö autor ei osalenud antud asukohapõhise tarkvara väljatöötamisel, kuid vormistas mudelipõhiste testide jaoks süsteemi WFM testitava alamosa spetsifikatsiooni.

Töö neljandas peatükis käsitletakse süsteemi WFM modelleerimist, mudeli ühendamist testitava süsteemiga ehk süsteemiadapteri ehitamist ning testide käivitamist.

Töö viiendas peatükis tehakse kokkuvõte läbiviidud testidest. Autor loetleb, millistele tulemustele testidega jõuti, ning annab hinnangu mudelipõhiste testide efektiivsusele antud süsteemi testimisel.

Töö juurde kuulub lisana süsteemi WFM mudeli ja testiadapteri kood ning selle juurde kuuluv dokumentatsioon.

Käesoleva magistritöö vahetulemusi tutvustas autor töötoas 20th Nordic Workshop on Programming Theory [4]. Lisaks publitseeriti konverentsi TESTCOM/FATES 2009 kogumikus süsteemi WFM testimise kogemustele tuginev artikkel, mis kirjeldab veebipõhiste süsteemide mudelipõhist testimist raamvaraga NModel [5].

Töö autor avaldab tänu firmale Regio, kelle loodud on testitav süsteem WFM, ja süsteemi WFM arhitektile Toivo Vajakale, kes aitas süsteemile esitatavate nõuete spetsifitseerimisel.

1. Mudelipõhine testimine

Käesolev peatükk annab ülevaate mudelipõhise testimise teooriast ning antud valdkonnas kirjutatud uurimistöödest.

1.1. Ülevaade mudelipõhisest testimisest

Kuna testimine kui oluline töövoog tarkvara arendusprotsessis on leidnud erialases kirjanduses laialdast käsitlemist, siis antud töö testimisel kui laiemal valdkonnal ei peatu. Olgu vaid meenutatud, et testimine on programmi käivitamine eesmärgiga leida vigu [1] ning testimise eesmärk on vigade avastamise kaudu tarkvara kvaliteeti tõsta.

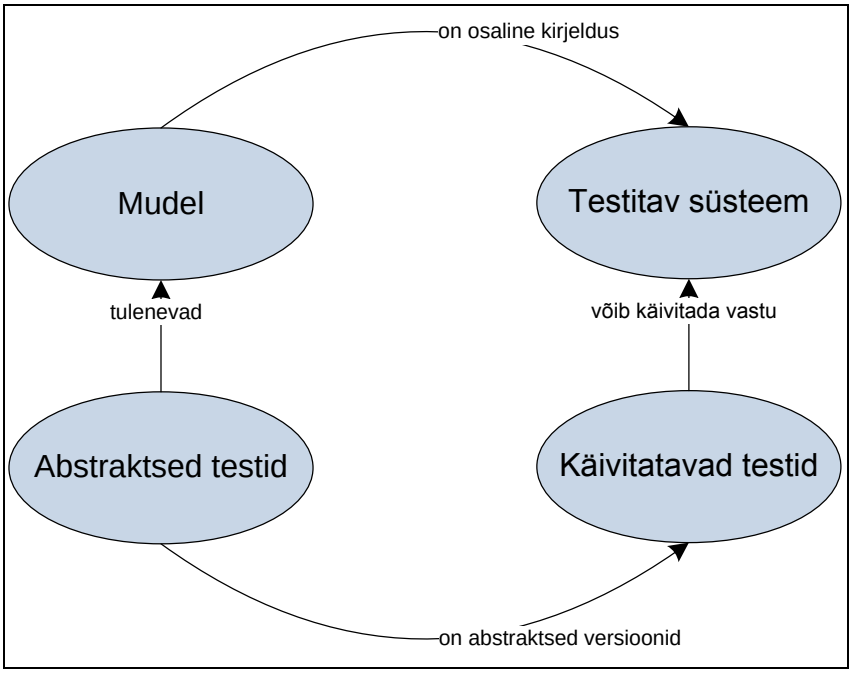
Mudelipõhine testimine on automaatse testimise meetod, kus testimise aluseks on süsteemi mudel [6]. Mudel on lihtsalt öeldes tarkvara käitumise kirjeldus, kus käitumist väljendatakse tarkvara lubatud sisendite, toimingute, üleminekutingimuste ja väljundite abil. Testimise käigus kontrollitakse, kas mudeli ja testitava süsteemi käitumised on üksteisele vastavad.

Enamasti on automaattestimise puhul kasutusel üpris piiratud hulk testlugusid ning testlugude loomine on ajamahukas töö. Mudelipõhine lähenemine annab võimaluse testlugusid mudelist automaatselt genereerida. Mudeli loomisele kulub küll mõningane aeg, kuid kui see on tehtud, saab kerge vaevaga genereerida suuremal hulgal testlugusid. Need võivad üksteisest mitmete omaduste poolest erineda – näiteks võivad testlood olla erineva pikkusega, katta erinevat funktsionaalsust või keskenduda ainult teatud tegevuste kordamisele. Selline lähenemine vastandub traditsioonilisele automaattestimisele, kus iga testlugu tuleb eraldi kodeerida. Seega on mudelipõhine testimine soovitatav olukordades, kus testide adekvaatse katvuse saavutamiseks on vajalik nii suur testlugude hulk, et nende käsitsi kirjutamine osutub väga suureks tööks.

Mudelipõhine lähenemine testimisele tagab selle, et nõuete muutumise korral tuleb muuta üksnes mudelit ning testid uuesti genereerida. Puudub vajadus testskripte ükshaaval muuta nagu traditsiooniliste automaattestimise meetodite puhul. Mudelipõhise lähenemise eeliseks on ka see, et mudeli loomine aitab leida vigu, vasturääkivusi ja mitmetimõistetavusi spetsifikatsioonis ja tõstab seega süsteemi loomise aluseks oleva spetsifikatsiooni kvaliteeti. Kui mudelipõhine testimine on haaratud tarkvaraarendusprotsessi, siis on soovitav mudeli kirjutamisega alustada võimalikult vara – seda saab teha enne tarkvara enda loomist – ja mudeli kirjutamine paneb

arendajaid spetsifikatsiooni varakult detailselt läbi mõtlema.

Mudelipõhise testimise põhimõisteid esitab joonis 1.



Joonis 1. Mudelipõhise testimise põhimõisted [7].

Mudel on testitava süsteemi osaline kirjeldus. Seega ei tähenda mudeli loomine, et sama süsteem realiseeritakse kahekordselt, vaid seda, et süsteemi oluline ja põhiline käitumine kirjeldatakse mudelis. Kuidas piiritleda süsteemi oluline ja põhiline käitumine, sõltub palju testitavast süsteemist ning mudelipõhise testimise eesmärkidest antud projektis.

Mudel luuakse lähtudes süsteemile esitatavatest nõuetest. Enamasti vaadeldakse testitavat süsteemi musta kastina, millele saadetakse sõnumeid ja millelt naasvate sõnumite järgi otsustatakse, millisesse olekusse süsteem jõudis. Vahel õnnestub mudelipõhises testimises ära kasutada süsteemi disainietapis loodud mudeleid, kuid enamasti tuleb mudeleid testimiseks sobivaks muutmisel oluliselt täiendada või päris otsast uuesti teha.

Mudelist tulenevad abstraktsed testid. Mudelist tuletatavad testid on funktsionaalsed testid, mis on samal abstraktsioonitasemel nagu mudel, kasutades mudelis defineeritud abstraktseid operatsioone ja väärtusi. Neid teste ei saa süsteemi peal otseselt käima lasta, kuna abstraktsioonitase ei ole selleks sobiv.

Abstraktsed testid on *käivitatavate testide* abstraktsed versioonid. Käivitatavad testid kirjeldavad täpselt, millisele tegevusele üks või teine mudelis määratletud üleminek vastab. Näiteks süsteemi sisse logimist kirjeldavate testide puhul võib abstraktne test öelda, et kasutaja peab õige kasutajanime ja salasõnaga süsteemi sisse pääsema, vale salasõna puhul aga peab ta saama veateate. Samas ei spetsifitseeri abstraktsed testid, millised kasutajanimi-salasõna-paarid on antud süsteemi puhul õiged ja millised mitte. Käivitatavad testid aga spetsifitseerivad täpselt, milliste kasutajanimede ja salasõnadega peab sisselogimine õnnestuma ja millistega mitte.

Käivitatavaid teste saab testitavas süsteemis tööle panna.

1.2. Mudelipõhiste testide käivitusmeetodid

Mudelipõhine testimine pakub kahte liiki võimalusi testide käivitamiseks.

- Esimesel juhul käivitatakse testimisvahend, nii et see läbib korraga nii mudelit kui ka testitavat süsteemi. Testlood luuakse jooksvalt ning käivitatakse kohe ning mudelit kasutatakse spetsifikatsioonile mittevastavuse tuvastamiseks. Antud testimisviisi nimetatakse käigupealt testimiseks.

- Teisel juhul käivitatakse testimisvahend esmalt selleks, et genereerida testlood. Seejärel saab nende testlugude abil testitavat süsteemi testida. Samu testlugusid saab süsteemil käivitada korduvalt. Antud testimisviisi nimetatakse eelsalvestatud testlugudega testimiseks või *offline*-testimiseks. Kui testitav süsteem saadab vastuseks mõne testloos ettenägematu vastuse, siis on loetakse test ebaõnnestunuks.

Mõlemal juhul on testlugude loomise ja vea tuvastamise aluseks mudel. Testimisvahend läbib mudelit ning loeb sellest välja, millised üleminekud on millistel tingimustel lubatud. Selle teadmise alusel saab testimisvahend kontrollida, kas analoogilised üleminekud on samadel tingimustel lubatud ka testitavas süsteemis.

Käigupealt testimist saab mõõta ajaühikutega, näiteks kui soovitakse süsteemi testida mingi ettenähtud aja jooksul. Võimalik on mõõta ja suunata ka testi katvust ehk seda, millistesse mudeli osadesse test süsteemi viib. Eelsalvestatud testlugudega testimine aga annab võimaluse täpselt samu teste mitu korda läbi teha.

Eelsalvestatud testlugudega testimise käigus loodavad testlood võib genereerida nii arvuti-loetaval kujul – nt klassid mingis programmeerimiskeeles – kui ka tavakeelsena – näiteks testlugude dokumendina teksti kujul.

Arvuti-loetavate testlugude eelsalvestamine tähendab, et mudelipõhine testimisvahend loob testlood mingisugusel arvutile sobival kujul ning testid saab läbi viia automaatselt. Arvuti-loetavaks kujuks võib olla näiteks kogum mingis programmeerimiskeeles defineeritud klasse, mis väljendavad genereeritud testimisloogikat. Samuti võib arvuti-loetav kuju olla defineeritud mõnes spetsiifiliselt testide kirjutamiseks mõeldud programmeerimiskeeles, näiteks TTCN-3 [8], [9].

Tavakeelsena esitatud testide eelsalvestamine tähendab, et mudelipõhine testimisvahend loob testlood inimloetavana ning testid saab hiljem käsitsi läbi viia. Testid võivad olla näiteks ingliskeelse dokumendina, kus on toodud testlugude sammud.

1.3. Mudeli ja testitava süsteemi vastavusseos

Mudelipõhise testimise eesmärk on kindlustada mudeli ja testitava süsteemi vaheline formaalne vastavus [10].

Nii mudelit kui ka testitavat süsteemi saab vaadelda liideseautomaadina. Liideseautomaat M koosneb järgmistest komponentidest:

- olekute hulk S;
- algolekute hulk - hulga S mittetühi alamhulk S^{init};
- juhitavate toimingute A^c ja jälgitavate toimingute A^o hulgad, mis on paarikaupa lõikumatud;
- üleminekufunktsioonid Γ^c ja Γ^o hulgast S vastavalt alamhulkadesse A^c ja A^o;
- üleminekufunktsioon δ, mis seob omavahel algse oleku ning selle oleku juures lubatud toimingu, mis viib järgmisse olekusse.

Vastavusseos mudeli ja testitava süsteemi vahel formaliseeritakse kui nende kahe automaadi vaheline täiendus (*refinement*), mis järgnevalt kirjeldatakse. Sümboliga M tähistatakse järgnevas spetsifikatsiooni liideseautomaati ning sümboliga N testitava süsteemi liideseautomaati.

Vahelduv simulatsioon ρ automaadist M automaati N on seos $\rho \in S_M \times S_N$, kus iga paari $(s, t) \in \rho$ kohta kehtivad väited:

1. $\Gamma^c_M(s) \subseteq \Gamma^c_N(t)$ ja $\Gamma^o_M(s) \supseteq \Gamma^o_N(t)$ ja
2. $\forall a \in \Gamma^o_M(s) \cup \Gamma^c_N(t), (\delta_M(s, a), \delta_N(t, a)) \in \rho$.

Siin tähistab esimene tingimus, et kõik mudelis defineeritud juhitavad toimingud on võimalikud ka testitavas süsteemis ning kõik võimalikud vastused, mille testitav süsteem võib anda, on mudelis lubatud. Teine tingimus tagab, et kui teatud algsete olekute paari korral on esimene tingimus täidetud, on esimene tingimus täidetud ka järgmises olekus, kuhu jõutakse mudelis lubatud juhitava toimingu või testitavas süsteemis lubatud jälgitava toimingu tulemusena.

Öeldakse, et liideseautomaat M täiendab liideseautomaati N, kui:

1. $A^o_M \supseteq A^o_N$ ja $A^c_M \subseteq A^c_N$ ja
2. leidub vahelduv simulatsioon ρ automaadist M automaati N, $s \in S^{init}_M$, $t \in S^{init}_N$, nii et $(s, t) \in \rho$.

Vahelduva täiendusseose mõte teiste sõnadega seisneb asjaolus, et igas võimalikus olekus, vaadeldes olukorda vaheldumisi juhitavate ja jälgitavate toimingute vaatevinklist, peavad ühest küljest olema testitavas süsteemis lubatud kõik mudelis lubatud juhitavad toimingud (ja võib-olla mõned veel, mida mudelis ei ole, st need, mida ei modelleeritud). Teisest küljest peavad olema mudelis lubatud kõik need jälgitavad toimingud, mida testitav süsteem vastavas olekus vastuseks pakub (ja võib-olla mõned veel).

Seose formaliseerimist kirjeldavad põhjalikumalt artiklid [10] ja [11].

1.4. Teised mudelipõhise testimise alased uurimistööd

Mudelipõhise testimise teemal on viimastel aastatel tehtud mitmeid uurimistöid ja avaldatud artikleid. Paljud neist piirduvad mingi metoodika kirjeldamisega, jõudmata selle rakendamiseni konkreetse tarkvara testimisel. Rida töid on aga ka juhtumianalüüsid, kus teatud metoodikat ja vahendit on rakendatud mõne programmi või komponendi testimiseks.

Heckel ja Lohmann kirjeldavad oma artiklis [12] metoodikat veebirakenduste mudelipõhiseks arenduseks ning testimiseks. Ühiste disainimustrite abil saab samu teste jooksutada nii kohalikus testikeskkonnas kui ka hajus-testikeskkonnas. Väliste süsteemide tööd saab testimisvahendiga vajadusel simuleerida. Artikkel piirdub metoodika kirjeldamisega, vahendit testimise läbiviimiseks siin välja ei töötatud.

Koopman, Plasmeijer ja Achten esitavad oma artiklis [13] automaatset mudelipõhist testimissüsteemi õhukese kliendiga veebirakenduste käigupealt testimiseks. Veebirakendused tuleb kirjeldada mittedeterministlikke laiendatud olekumasinaid kasutades. Laiendatud olekumasin võimaldab traditsioonilise olekumasinaga võrreldes üleminekuid väljendada *if*-lausetena, mis koosnevad hulgast tingimustest. Üleminek saab toimuda, kui kõik lauses olevad tingimused on täidetud. Testisüsteem realiseeriti programmeerimiskeeles Clean. Clean on funktsionaalne programmeerimiskeel, mis on suuresti mõjutatud programmeerimiskeelest Haskell. Testide käivitamiseks kasutatakse vahendit $G\forall ST$. Artikkel kirjeldab kahte juhtumianalüüsi, kus viiakse läbi sünkroonseid teste: vahendi $G\forall ST$ abil antakse testitavale süsteemile sisend ning kontrollitakse vastuseks tulnud HTML-lehte. Etteruttavalt olgu öeldud, et süsteemi WFM testimisel vahendiga NModel loodi lisaks sünkroonsetele testidele ka asünkroonsed testid, kus teatud toiminguid algatas mitte testimisvahend, vaid testitav süsteem.

Xie ja Memon kirjeldavad oma artiklis [14] automaatseid testioraakleid tarkvara kasutajaliidesepõhiseks testimiseks. Testimises osalevad sündmused peavad olema deterministlikud. Kirjeldatud metoodika ja kasutatud tarkvaralised vahendid näevad ette, et kasutajaliides tuleb modelleerida kasutajaliidese komponentide ja nende omaduste abil. Uuringus arendati kuus erinevat testioraaklit ning neid rakendati eksperimendis, et oraakleid omavahel võrrelda. Uuring näitas, et testioraakli loomisel tehtud otsused sellest, mida ja kui tihti testi käigus kontrollida, mõjutavad olulisel määral testimise hinda ning efektiivsust.

Baresi, Fraternali, Tisi ja Morasca keskenduvad artiklis [15] veebirakenduste generaatori mudelipõhisele testimisele. Nimetatud töös asendati veebirakenduste käsitsikirjutamine koodi

automaatse genereerimisega ning seega suunati ka testimise fookus individuaalsete veebirakenduste testimiselt generaatori testimisele.

Projektis D-MINT [16] osales 25 partnerit kuuest Euroopa riigist. Projekti eesmärgiks oli mudelipõhise testimise meetodite ja tööriistade rakendamine tööstuslike testiülesannete lahendamiseks. Hinnati mudelipõhise tehnoloogia hetkeseseisu tööstuslike vajaduste rahuldamiseks ja oodati tööstusest tagasisidet meetodite ja tööriistade edasiarenduseks. Kogu projekt baseerus juhtumianalüüsidel, mis olid võetud erinevatest tööstusvaldkondadest. Üheks uurimissuunaks projektis oli Tartu tänavavalgustuse kontrolleri mudelipõhine testimine. Tegemist oli juhtumianalüüsiga, milles tehniliste vahenditena mudelipõhiste testide tegemisel kasutati vahendeid Poseidon (modelleerimine), MOTES (TTCN-3-testide genereerimine mudelist) ja MessageMagic (TTCN-3-testide jooksutamiseks). Nimetatud juhtumianalüüsis oli suureks väljakutseks adapteri ehitamine, mis suhtles tänavavalgustuse kontrolleriga. Käesolevas magistritöös kulus mudeli ja adapteri loomisele umbkaudu võrdne aeg, kuid projektis D-MINT oli adapteri ehitamine selgelt suurema keerukusega kui mudeli loomine. Adapteri ehitamise keerukus projektis D-MINT tuleneb kontrolleri liidesest välismaailmaga, mis sisaldab analoog- ja digitaalseid elektrilisi signaale. Süsteemiadapteri ülesanne on testisüsteemi sõnumite tõlkimine elektrilisteks signaalideks ja vastupidi.

Vahendeid, mis toetavad mõne programmeerimiskeele (nagu näiteks C#) asemel skriptikeelt TTCN-3, on mitmeid, samuti on TTCN-3-testimist rakendatud veebirakenduste testimisel [17], [18].

Grieskamp, MacDonald, Kicillof, Stobie, Wurden ja Nandan käsitlevad oma artiklis [19] tarkvara dokumentatsiooni testimist, sealhulgas mudelipõhiselt. Mudelipõhise testimise vahendina on kasutusel Spec Explorer 2007.

Javed, Strooper ja Watson demonstreerivad oma artiklis [20] meetodit, mis võimaldab modelleerimiskeeles UML kirjeldatud järgnevusdiagrammidest automaatteste genereerida.

Antud meetodit järgides ehitasid artikli autorid näidisvahendi pangaautomaadi simulaatori testimiseks.

Andres Kulli doktoritöö [2] kirjeldab reaktiivsete süsteemide mudelipõhiseks testimiseks loodud tehnoloogiat. Antud tehnoloogia loomisel olid põhilisteks eesmärkideks see, et see oleks tavaliste testiinseneride jaoks piisavalt lihtne kasutada ning et see sobiks ka tööstuslike testimisülesannete lahendamiseks. Testimisvahendiks, mida antud doktoritöö kirjeldab, on MOTES [16]. Loodud tehnoloogiat illustreeritakse kahe juhtumianalüüsiga.

Jääskeläinen, Katara, Kervinen, Heiskanen, Maunumaa ja Pääkkönen kirjeldavad oma artiklis [3] teenust nutitelefonide rakenduste testimiseks. Autorid toovad välja probleemi, et vaatamata mudelipõhise lähenemise teoreetilistele eelistele traditsioonilise automaattestimise ees on mudelipõhine testimine tööstuses vähe rakendust leidnud. Seetõttu tutvustatakse artiklis arendatavat lahendust, mis on ette nähtud spetsiifiliselt nutitelefonidele, täpsemalt sellistele, mille operatsioonisüsteemiks on Symbian S60. Autorid usuvad, et spetsiifiline lahendus on tööstuses kergemini rakendatav kui üldine mudelipõhise testimise raamistik.

Käesolevat magistritööd eristab mitmetest teistest mudelipõhise testimise alastest töödest see, et antud töös loodi asünkroonne testivahend ja testivahend toetab mittedeterminismi. Samuti kasutatakse antud töös peamiselt käigupealt testimist, samas kui sageli kasutatakse mudelipõhise testimise rakendamisel testide eelsalvestamist ning seejärel käivitamist.

2. Mudelipõhise testimisvahendi valik

Antud peatükis valitakse saadaolevate mudelipõhise testimise tööriistade seast välja vahend, mida käesolevas töös kasutada. Ühtlasi kirjeldatakse valitud vahendit lähemalt.

2.1. Vahendi valiku kriteeriumid

Käesolevas töös pakkus huvi veebisüsteemide testimine. Katseobjektiks valitud süsteem WFM, mida kirjeldatakse detailselt järgmises peatükis, koosneb veebiserverist ning rakendusserverist, kusjuures rakendusserver suhtleb omakorda veel mitme komponendiga, st süsteem WFM omab peale veebiliidese veel mitut liidest alamsüsteemidega suhtlemiseks.

Süsteemi WFM oli varasemalt testitud järgmiselt:

- funktsionaalsed testid – viidi läbi käsitsi veebiliidese kaudu;
- koormustestid – vahendiga JMeter [21] viidi läbi automaattestid, mis imiteerisid veebiklienti.

Need testid vaatlesid süsteemi WFM üksnes veebiliidese kaudu. Samas on huvipakkuv ka kontroll, kas süsteemi WFM töös jõuavad andmed korrektselt teistesse alamsüsteemidesse, millega süsteem WFM suhtleb. Ühtlasi koosnesid käsitsitestid ja vahendiga JMeter läbi viidud testid piiratud hulgast stsenaariumidest.

Mudelipõhise testimisvahendi valikul eeldati, et see peab toetama mittedeterminismi ehk oskama adekvaatselt käituda olukorras, kus ühele sisendsõnumile vastab mitu väljundsõnumit ja väljundsõnumist sõltub süsteemi edasine käitumine. Ühtlasi eeldati testimisvahendilt asünkroonsuse tuge, kuna oli huvi kontrollida sõnumite kohalejõudmist teistesse alamsüsteemidesse, millega süsteem WFM suhtleb. Soovitav oli, et tegemist oleks vabavaralise vahendiga. Vähemoluliseks kriteeriumiks oli lähtekoodi avatus. Eelistatumad olid vahendid, mis võimaldasid modelleerida mõnes laiatarbekeeles ja mille kohta oli kättesaadav detailne dokumentatsioon.

2.2. Mudelipõhise testimise vahendid

Mudelipõhist testimist toetavaid vahendeid on maailmas loodud mitmeid. Ülevaate paljudest neist annab raamat [6]. Vahendeid, mis tundusid antud töös valikut tehes huvipakkuvad, esitab

tabel 1. Tabelis on esitatud omadused, mis olid vahendite valimisel olulised, ning milles esines vahendite vahel erinevusi.

Tulbas *Tasuta* tähistab märge +/- seda, et vahendit on lubatud tasuta kasutada ainult teadustöös, aga mitte kommertseesmärkidel. Kui vahend on tasuta, ei tähenda see ilmtingimata avatud lähtekoodi.

Vahendi nimetus	Keeled	Mittedeter-minismi tugi	Asünkroonsuse tugi	Tasuta
Qtronic [22]	UML, Java, C#	+	+	-
Uppaal-Tron [23]	Uppaal	+	+	+/-
Spec Explorer 2006* [24], [25]	AsmL, Spec#	+	+	+/-
NModel [26]	C#	+	+	+
MOTES** [16]	UML, TTCN-3	+	+	+
MessageMagic [16]				-
TorX [27]	Promela	+	-	+
TGV [28]	LOTOS, SDL	-	-	+/-

Tabel 1. Mudelipõhise testimise vahendeid.

* Käesolevat tööd alustades oli kättesaadav vahendi Spec Explorer versioon 2006, mis võimaldab mudeleid kirjutada keeltes AsmL ja Spec#. Hiljem on Microsoft Visual Studio 2010 lisamoodulina välja tulnud Spec Explorer 2010, mis lubab kasutada raamistiku .Net keeli.

** Vahendi MOTES sisendiks olevate mudelite loomiseks on vajalik tasuline tarkvara PoseidonUML [29]. Vahendeid MOTES ja MessageMagic vaadeldakse ühise mudelipõhise testimise vahendite komplektina.

Mitmete omaduste poolest olid vaadeldud vahendid sarnased. Näiteks toetavad mudelipõhise testimise vahendid reeglina nii käigupealt kui ka eelsalvestatud testlugudega testimist. See kehtib ühtlasi kõigi nende vahendite kohta, mida esitab tabel 1.

Samuti toetavad tööriistad, mida loetleb tabel 1, vähemal või rohkemal määral kompositsionaalset modelleerimist. Nimelt saab enamikes modelleerimiskeeltes mudelid mingil moel tükkideks jagada – näiteks eraldi automaatideks, mis on omavahel teatud viisil

sünkroniseeritud. Vahend QTronic võimaldab kompositsiooni eraldi automaatideks või Java komponentideks. Tööriist Uppaal-Tron võimaldab kompositsiooni eraldi automaadimallidest. Vahend TorX lubab modelleerida keeles Promela, mis võimaldab samuti defineerida eraldi automaate. Tööriista TGV puhul on samuti kompositsioon seotud modelleerimiskeeltes võimaldatavaga. Raamvara NModel võimaldab lisaks toimingute kaupa organiseeritud kompositsioonile ka kompositsiooni toimingu osade kaupa. See tähendab võimalust lisada mudelisse ükshaaval sisse- ja väljalülitatavaid osi, mis seavad olemasolevatele toimingutele lisapiiranguid. Näiteks võib sisselogimist kirjeldavale toimingule eraldi lisada piirangu, et sisse logida püütaks ainult korrektse kasutajanime ja salasõnaga.

Eespool kirjeldatud vahendite hulgast tundus sobivaim raamvara NModel. Tabel 1 näitab, et vaatluse all olnud vahenditest on see ainuke, millel on kõik järgmised omadused: see on tasuta, toetab mittedeterminismi ja asünkroonsust ning puudub vajadus mõne uue, vähem levinud keele õppimiseks.

NModel on mitmetelt omadustelt sarnane tööriistaga QTronic, kuid vabavaraline. Nagu tabel 1 näitab, võimaldab raamvara NModel erinevalt mitmetest teistest nimetatud vahenditest modelleerimist laiatarbekeeles (C#), mistõttu puudus vajadus uue keele õppimiseks. Selgeks tuli õppida vaid raamvara NModel API ja andmestruktuurid. Lisaks piisab raamvara NModel puhul testide jooksutamiseks mõne megabaidi suurusest failide komplektist, samas kui näiteks vahendi Spec Explorer uuema versiooni puhul tuleb käivitada kogu Visual Studio. Raamvara NModel eeliseks mitmete teiste tööriistade ees on ka avatud lähtekood. Lisaks on raamvara NModel kohta olemas detailne dokumentatsioon [26], [30].

Nimetatud eelistest lähtuvalt valiti antud töös kasutamiseks välja raamvara NModel.

2.3. Raamvara NModel

Raamvaral NModel on järgmised omadused:

- tegemist on avatud lähtekoodiga tarkvaraga;
- NModel toetab kompositsionaalset modelleerimist (mudeli jagamine üksusteks).

- NModel toetab abstraktseid andmetüüpe nagu hulgad ja kujutised (*Set* ja *Map*).

- NModel toetab asünkroonsete testide loomist ehk olukorda, kus osa toiminguid ei ole testimisvahendi-juhitavad, vaid üksnes jälgitavad.

- NModel toetab „õhukest" mittedeterminismi [30] ehk olukorda, kus ühele sisendsõnumile võib vastuseks tulla mitu erinevat väljundsõnumit.

Raamvara NModel ning selle rakendamise kirjeldamisele on pühendatud raamat [30].

Raamvara NModel sisaldab järgmisi vahendeid.

- Atribuudid ja andmetüübid mudelprogrammide kirjutamiseks programmeerimiskeeles C#. Enim kasutatavad atribuudid on [Action], mis defineerib mudelis ülemineku ühest olekust teise, ning [Feature], mis võimaldab mudeli üles ehitada ükshaaval sisse ja välja lülitatavatest komponentidest. Enim kasutatavad andmetüübid on *Set,* mis defineerib hulga, ning *Map,* mis defineerib kujutise ehk võti-väärtus-paarid.

- Visualiseerimis- ja analüüsivahend MPV (Model Program Viewer), mis võimaldab C# koodina kirjeldatud mudeleid visuaalselt kujutada.

- Olekute saavutatavuse kontrolli vahend MC (Model Checker), mis lubab kokku lugeda mudeli olekuid ja üleminekuid ning kontrollida, kas mingi etteantud olek on saavutatav või mitte.

- Testide genereerimisvahend OTG (Offline Test Generaator), mis valmistab ette testlood. Neid testlugusid on võimalik käivitada raamvara NModel vahendiga CT.

- Testide käivitamise vahend CT (Conformance Tester), mis võimaldab teha nii käigupealt testimist kui ka käivitada eelsalvestatud testlugusid.

NModel võimaldab mudelprogramme luua ja kasutada kahel moel:

- koondmudelprogramm (*contract model program*) esitab süsteemi kõiki lubatud käitumisi,

- stsenaariummudelprogramm (*scenario model program*) esitab ainult hulka tegevuste

jadasid, mis on omavahel mõne tunnuse alusel seotud.

Antud töös vaadeldakse nii koond- kui ka stsenaariummudelprogramme.

Koondmudelprogramm on esitatav süsteemi erinevate osade kirjelduste kompositsioonina. Kompositsioon on matemaatiliselt kirjeldatud artiklis [31].

2.4. Lihtne näidis-mudelprogramm

Järgnevalt kirjeldatakse mudelite loomist raamvara NModel abil ühe äärmiselt lihtsa näite alusel. Näide annab lugejale ettekujutuse põhimõttest, kuidas mudelprogrammide loomine raamvara NModel abil toimub, ning annab raamvarast NModel esmase ülevaate. Raamvara NModel rakendusvõimalusi illustreeritakse antud töös ka edaspidi, kui kirjeldatakse süsteemil WFM läbi viidud teste.

Olgu testitav süsteem selline, kuhu kuuluvad elektripirn ning sisse- ja väljalülitamist võimaldav lüliti. Süsteemis kehtivad järgmised reeglid.

1. Kui kasutaja vajutab nuppu „On“, läheb pirn põlema.
2. Kui kasutaja vajutab nuppu „Off“, pirn kustub.
3. Nuppu „On“ on võimalik vajutada siis ja ainult siis, kui pirn ei põle.
4. Nuppu „Off“ on võimalik vajutada siis ja ainult siis, kui pirn põleb.

Eelnevas loetelus tähistavad punktid 1–2 toiminguid ning punktid 3–4 vastavalt nende toimingute üleminekutingimusi.

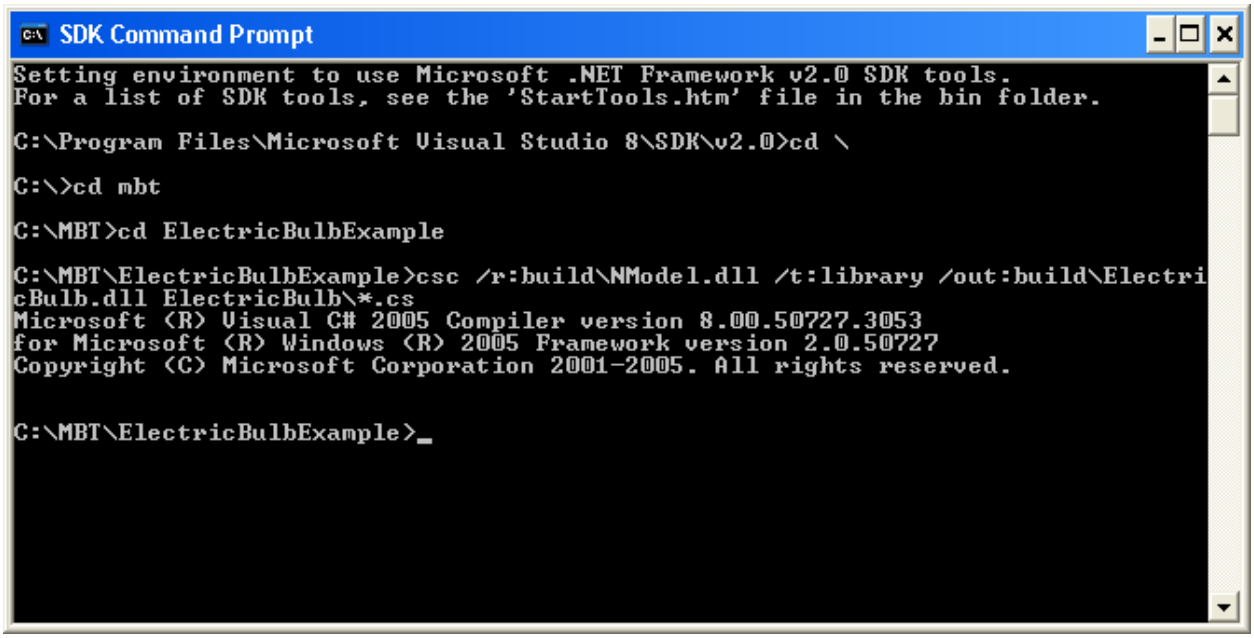
Eelkirjeldatud süsteemi modelleerimiseks luuakse C# koodifail, mille sisu kujutab joonis 2.

```
using System;
using NModel;
using NModel.Attributes;
using NModel.Execution;
using NModel.Terms;

namespace ElectricBulb
{
    [Feature("SwitchBulbOnAndOff")]
    public static class Contract
    {

        #region toimingud ja üleminekutingimused
        //Näitab, kas pirn parasjagu põleb (true) või ei põle (false).
        private static bool bulbIsOn = false;

        [Requirement("1. Kui kasutaja vajutab nuppu 'On', läheb pirn põlema.")]
        [Action]
        //Toiming SwitchOn määratleb, mis toimub, kui kasutaja vajutab nuppu 'On' - siis läheb pirn põlema.
        public static void SwitchOn() {
            bulbIsOn = true;
        }

        //Toimingu SwitchOn üleminekutingimus - pirni on võimalik sisse lülitada
        //siis ja ainult siis, kui pirn ei põle.
        public static bool SwitchOnEnabled() {
            return (!bulbIsOn);
        }

        [Requirement("2. Kui kasutaja vajutab nuppu 'Off', pirn kustub.")]
        [Action]
        //Toiming SwitchOff määratleb, mis toimub, kui kasutaja vajutab nuppu 'Off' - siis pirn kustub.
        public static void SwitchOff() {
            bulbIsOn = false;
        }

        //Toimingu SwitchOff üleminekutingimus - pirni on võimalik välja lülitada
        //siis ja ainult siis, kui pirn põleb.
        public static bool SwitchOffEnabled()
        {
            return (bulbIsOn);
        }

        #endregion

        konstruktorid

    }
}
```

Joonis 2. Elektripirni sisse- ja väljalülitamist kirjeldav mudel keeles C#.

Nagu koodinäidises näha, kirjeldati mudelis mõlemad toimingud (nupu „On“ vajutamine ning nupu „Off“ vajutamine) ning üleminekutingimused, mis näitavad, millal kumbagi toimingut on võimalik läbi viia.

Mudel kompileeritakse käsurealt .NET SDK vahendi csc abil. Mudeli kompileerimist illustreerib joonis 3.



Joonis 3. Mudeli kompileerimine.

Kompileeritud mudel kujutab endast raamistiku .NET dünaamiliselt lingitavat teeki ehk DLL-faili. Seda faili kasutavad raamvara NModel vahendid mudeli analüüsiks ja testide genereerimiseks.

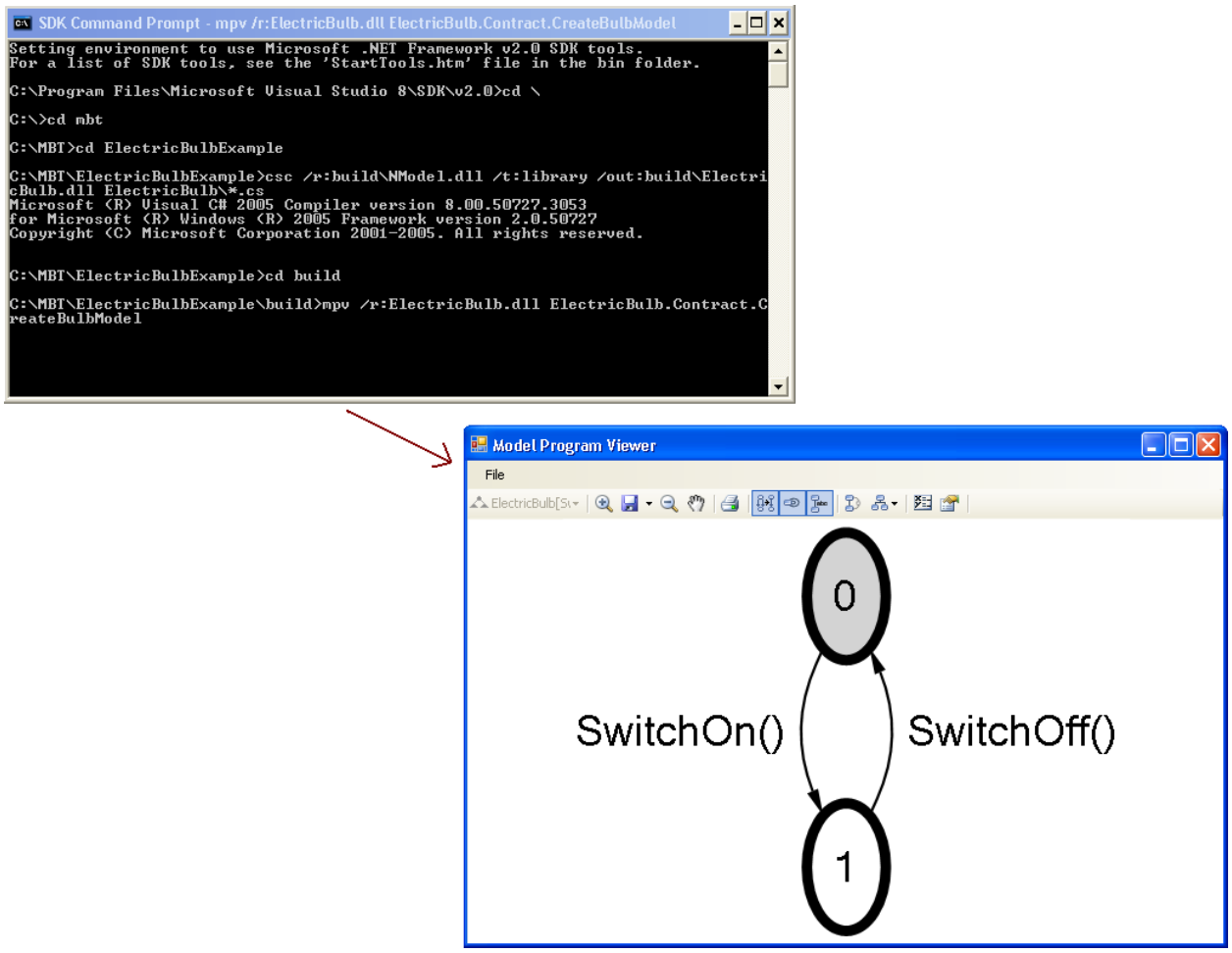
Eelnevast koodinäidisest oli loetavuse huvides välja jäetud konstruktorite alamjaotus, kuna toimingute ja üleminekutingimuste vaatlemisel ei ole see oluline. Antud lihtsa näidise puhul on mudelil ainult üks konstruktor – selline, mis loob antud mudeli täies ulatuses. Konstruktor on näha koodinäidises, mida kujutab joonis 4. Kui eelmises koodinäidises oli ahendatud alamjaotus "konstruktorid", siis antud koodinäidis on samast failist, kuid siin on selguse huvides ahendatud alamjaotus "toimingud ja üleminekutingimused", kuna see ei ole konstruktorite vaatlemisel oluline.

```
using System;
using NModel;
using NModel.Attributes;
using NModel.Execution;
using NModel.Terms;

namespace ElectricBulb
{
    [Feature("SwitchBulbOnAndOff")]
    public static class Contract
    {

        toimingud ja üleminekutingimused

        #region konstruktorid

        //Konstruktor, mis loob pirni sisse- ja väljalülitamist kujutava mudeli.
        public static ModelProgram CreateBulbModel()
        {
            return new
                    LibraryModelProgram(typeof(Contract).Assembly, "ElectricBulb",
                    new Set<string>("SwitchBulbOnAndOff"));
        }
        #endregion

    }
}
```

Joonis 4. Elektripirni sisse- ja väljalülitamist kirjeldava mudeli konstruktor.

Mudelit saab visuaalselt kujutada raamvarasse NModel kuuluva vahendi MPV abil. Mudeli visuaalset esitamist illustreerib joonis 5.

Mudeli kujutamine visuaalselt annab mudelprogrammi loojale esmase vahendi mudeli korrektsust hinnata. Visualiseeritud mudelit saab kasutada mudelprogrammi silumisel, kuna seal on näha võimalikud probleemsed kohad mudelis: ebaõiged üleminekud, tupikud ning tsüklid.

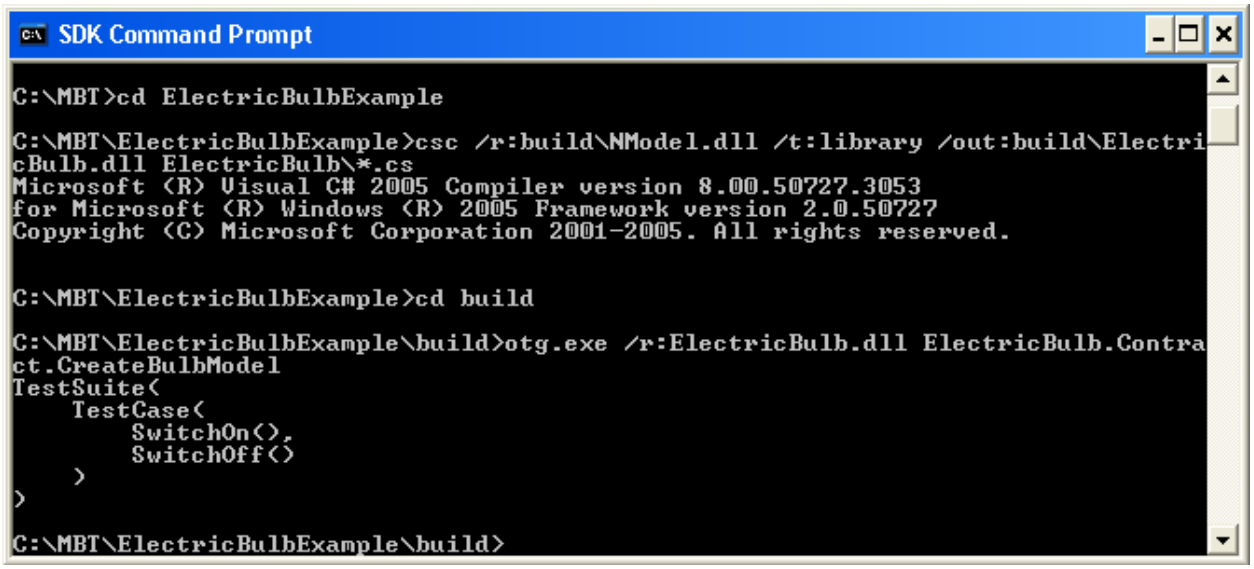


Joonis 5. Mudeli visuaalne esitamine.

Raamvarasse NModel kuuluv vahend OTG võimaldab mudeli alusel genereerida testlood. Selleks uurib vahend OTG mudelit ning salvestab rea mudelit läbivaid teid faili. Selline mudelit läbivate teede kogumik kujutab endast testlugude komplekti ning iga tee kujutab endast testlugu. Vaikimisi püüab vahend OTG luua testid, mis katavad süsteemi kõik üleminekud. Seda, kui palju ja kuidas testid süsteemi katavad, on võimalik reguleerida, andes vahendile OTG ette stsenaariumi.

Testide genereerimisel arvestab vahend OTG, et test peab lõppema lubatud olekus. Mudeli koodis saab lubatud lõppolekud defineerida, kasutades atribuuti [AcceptingStateCondition], mis määratleb mudeli olekumuutujate lubatud väärtused testi lõppedes. Kui mudelis ei ole lubatud lõppolekuid defineeritud, loeb testimisvahend korrektseks testi lõppemise mistahes olekus.

Kirjeldatud lihtsa näite puhul tuleb testlugude hulk väga väike. Testlugude genereerimist illustreerib joonis 6.



Joonis 6. Testlugude genereerimine mudeli alusel.

Antud peatükis ei demonstreerita testide käivitamist vahendi Conformance Tester abil. Testide käivitamiseks peab olemas olema reaalne süsteem, mida testida, ja adapter, mis ühendab keeles C# kirjeldatud mudeli ning testitava süsteemi. Testide käivitamist vahendi Conformance Tester abil demonstreeritakse asukohapõhise süsteemi WFM testide kirjeldamisel.

3. Asukoha määramise süsteem WFM

Antud peatükk kirjeldab asukoha määramise süsteemi WFM, mis on käesoleva töö testimise objekt. Töö autor ei osalenud süsteemi WFM väljatöötamises, kuid vormistas süsteemi testitavale funktsionaalsusele esitatavad nõuded ning formaliseeris need raamvara NModel abil mudeliks.

Süsteem WFM valiti katseobjektiks, kuna autori tööandjas tekkinud huvi mudelipõhise testimise vastu langes kokku süsteemi WFM valmimisajaga. WFM sobib katseobjekti rolli, kuna see koosneb paljudest omavahel nii sünkroonselt kui asünkroonselt sõnumeid vahetavatest komponentidest, mille interaktsiooni kirjeldatakse nõuete spetsifikatsiooniga.

3.1. Ülevaade asukoha määramise süsteemist WFM

Süsteem WFM on asukohapõhine rakendus, mis võimaldab jälgida töötajate liikumist ning asukohta, töötajatele sõnumeid saata ning töötajate liikumise ajalugu vaadata. Süsteem on ette nähtud asutustele, kellel on oma töös vaja töötajate liikumist jälgida ning planeerida. Süsteem WFM kasutab mobiilpositsioneerimist, andes asukoha täpsuseks linnas keskmiselt paarsada meetrit, maapiirkondades aga keskmiselt paar kilomeetrit. Süsteemi WFM kasutajaliides on veebipõhine.

3.2. Kasutajarollid asukoha määramise süsteemis WFM

Süsteemi WFM kasutajarolle esitab tabel 2.

Roll	**Selgitus**
Super-administraator	Süsteemi WFM peakasutaja. Saab hallata *asutusi* (*company*). Asub mobiilioperaatori juures, kes oma klientidele süsteemi WFM teenust pakub.
Asutuse administraator	Süsteemi WFM peakasutaja asutuses (nt kullerfirmas).

Roll	Selgitus
Asutuse operaator	Süsteemi WFM kasutaja asutuses. Tegeleb otseselt tööde planeerimise ning logistikaga ning saab töötajaid positsioneerida ning neile tööülesandeid jagada.
Asutuse töötaja	Isik, keda asutuse operaator positsioneerida saab ning kellele tööülesandeid jagab. Kasutab mobiiltelefoni, mille abil teda positsioneeritakse ning kuhu lühisõnumiga (SMS) tööülesandeid saadetakse. Ei kasuta süsteemi WFM veebiliidest.

Tabel 2. Kasutajarollid süsteemis WFM.

Mudelipõhistes testides vaadeldi olukorda, kus *asutuse operaator* kasutab süsteemi WFM veebiliidest ning positsioneerib *asutuse töötajat*. *Super-administraatori* ning *asutuse administraatori* rolli kasutati ainult vajaliku keskkonna ettevalmistamisel – test-asutuste, -operaatorite ning -kasutajate loomiseks süsteemis WFM.

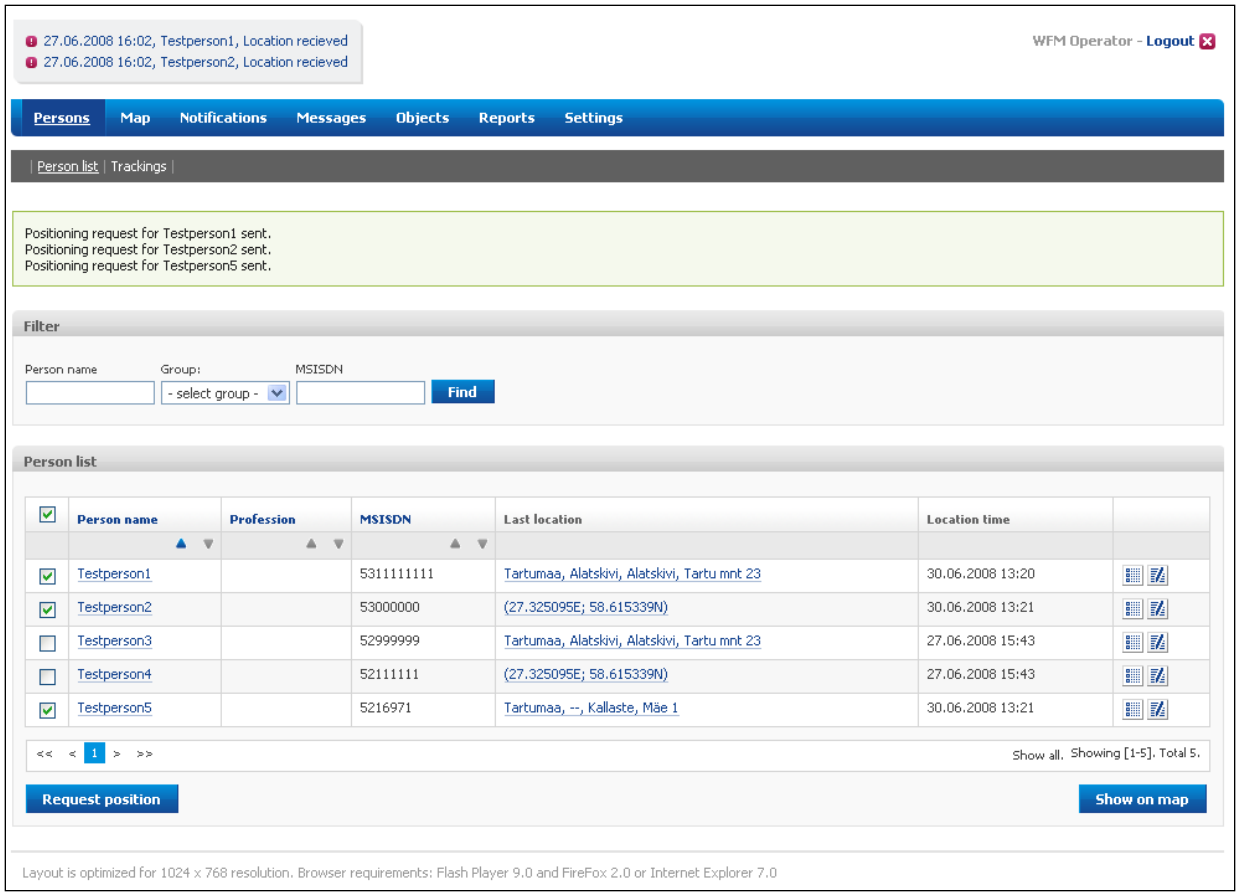
3.3. Asukoha määramise süsteemi WFM kasutajaliides

Illustreerimaks teste, vaadeldakse järgnevalt, kuidas tegevused, mida mudelipõhise testimisvahendiga imiteeriti, paistavad süsteemi WFM kasutajaliideses.

Süsteemi WFM sisselogimiseks kasutatakse lehte, mida kujutab joonis 7.

Joonis 7. Süsteemi WFM login-aken.

Töötajate nimekirja vaatamiseks, sealt positsioneeritavate valimiseks ning asukohtade vaatamiseks teksti kujul kasutatakse töötajate nimekirja lehte, mida kujutab joonis 8.

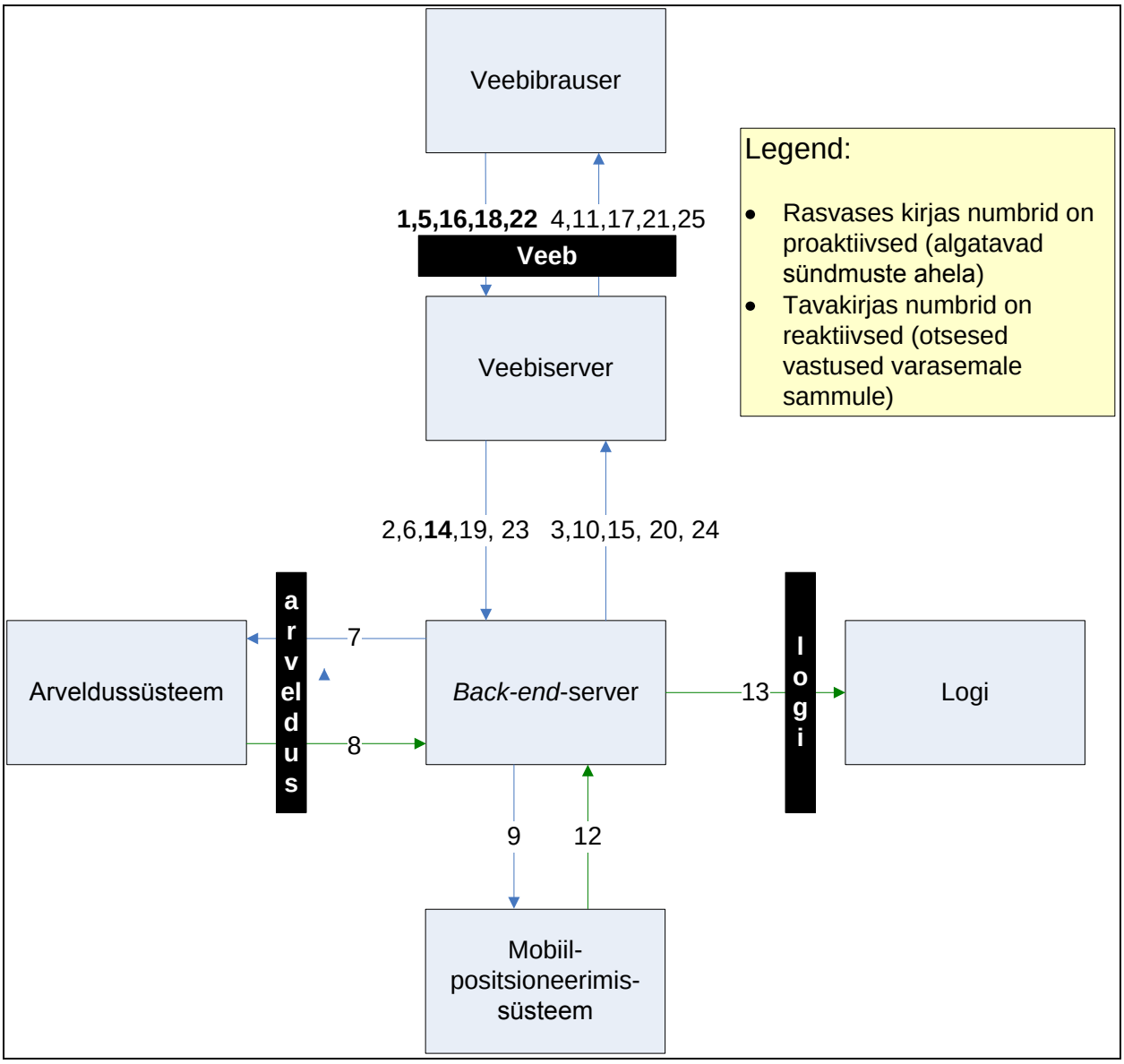


Joonis 8. Töötajate nimekirja leht süsteemis WFM.

Antud töös testitav funktsionaalsus on lõppkasutaja jaoks kaetud nende kahe lehega. Ülejäänud lehti, näiteks neid, kus asutusi ja kasutajaid defineerida saab, antud töös ei vaadelda, kuna neid lehti mudelipõhiselt ei testitud. Testimiseks valiti kriitilisim osa süsteemi WFM funktsionaalsusest – need tegevused, mida kasutajad kõige sagedamini läbi viivad.

3.4. Testitavale funktsionaalsusele esitatavad nõuded

Süsteemi testitavat funktsionaalsust kirjeldab skemaatiliselt joonis 9. Joonisel on heledate kastidena kujutatud süsteemi komponendid. Mustade kastidega on näidatud liidesed, mis mudelipõhise testimise käigus mudelipõhise testimisvahendi külge ümber lülitati: nii veebiklienti, arveldus- kui logisüsteemi pandi imiteerima testimisvahend. Nooled näitavad sõnumeid süsteemi komponentide vahel ning numbrid nooltel sõnumite liikumise järjekorda.



Joonis 9. Sündmuste järgnevus süsteemis WFM.

Süsteemi funktsionaalsusele esitatakse järgmised nõuded. Joonis 9 esitab nõuete konteksti süsteemis.

Kasutajaroll: asutuse operaator.

1. Kasutaja logib süsteemi WFM sisse. Veebibrauser saadab veebiserverile sisselogimispäringu, mis sisaldab kasutajanime ja salasõna.
2. Veebiserver saadab sisselogimispäringu edasi *back-end*ile.

3. *Back-end* saadab veebiserverile vastuse – kas antud sisselogimiskatse õnnestus.
4. Veebiserver edastab vastuse veebibrauserile ning eduka sisselogimise korral kuvatakse kasutajale töötajate nimekirja leht. Ebaõnnestunud sisselogimise korral kuvatakse kasutajale uuesti *login*-aken ning veateade.
5. Kasutaja käivitab süsteemi WFM avalehel töötajate positsioneerimise. HTTP-päring, mille veebibrauser veebiserverile saadab, sisaldab positsioneeritavate isikute järjekorranumbreid (0..*n*).
6. Veebiserver saadab positsioneerimispäringu *back-end*i.
7. *Back-end* pöördub arveldussüsteemi poole, et operaatorilt teenuse eest raha võtta. Negatiivset juhtumit, kus arveldussüsteemis mingil põhjusel raha maha võtmine ei õnnestu, antud mudelipõhistes testides ei vaadelda.
8. Arveldussüsteem saadab *back-end*ile vastuse teenuse eest tasumise õnnestumise kohta.
9. *Back-end* saadab positsioneerimispäringu asünkroonselt mobiilpositsioneerimise süsteemi.
10. *Back-end* saadab veebiserverile vastuse, et positsioneerimist on alustatud.
11. Veebiserver saadab päringu vastuse edasi veebibrauserisse ning kasutaja näeb seda ekraanil. Päringu vastuseks ei ole veel töötaja asukoht, vaid positiivsel juhul teade, et positsioneerimist on alustatud, negatiivsel juhul süsteemne veateade.
12. *Back-end* saab mobiilpositsioneerimise süsteemilt positsioneerimise vastuse. Positiivsel juhul on selleks töötaja asukoht, probleemi korral veateade.
13. *Back-end* kirjutab info positsioneerimise toimumisest logisse.
14. Veebiserver küsib ettenähtud ajavahemiku tagant *back-end*ilt, kas asutuse töötajate asukohaandmetes on muudatusi ehk kas uued positsioneerimise tulemused on saabunud.
15. *Back-end* tagastab veebiserverile vastuse (jah või ei), kas uued positsioneerimise tulemused on saabunud.
16. Veebibrauser küsib ettenähtud ajavahemiku tagant veebiserverilt, kas veebiserveris on uusi asutuse töötajate asukohaandmeid. Seda teeb veebibrauser alati, kui kasutaja on töötajate nimekirja lehel, hoolimata sellest, kas antud kasutaja algatas antud sessiooni käigus positsioneerimise või mitte. Seda selletõttu, et sama asutuse töötajate

positsioneerimise võib käivitada ka mõni teine kasutaja ja sama asutuse mistahes kasutaja käivitatud positsioneerimise tulemused ilmuvad siin aknas nähtavale.

17. Veebiserver tagastab veebibrauserile loenduri väärtuse: 0 – uusi positsioneerimistulemusi ei ole laekunud; 1 – uued positsioneerimistulemused on olemas. Kui uusi tulemusi pole, liigub protsess tagasi sammu 16. Kui uued tulemused on olemas, liigub protsess sammu 18.
18. Veebibrauser küsib üle HTTP töötajate viimased positsioneerimistulemused.
19. Veebiserver saadab päringu *back-end*i, küsides sealt töötajate uusimaid teadaolevaid positsioneerimistulemusi.
20. *Back-end* tagastab töötajate uusimad teadaolevad positsioneerimistulemused.
21. Veebiserver tagastab XML-kujul vastuse, mis sisaldab antud asutuse töötajate kahte viimast positsioneerimistulemust.
22. Operaator logib süsteemist WFM välja. Veebibrauser saadab veebiserverile vastava päringu.
23. Veebiserver saadab väljalogimis-päringu edasi *back-end*ile.
24. *Back-end* saadab veebiserverile vastuse – kas antud väljalogimine õnnestus.
25. Veebiserver edastab vastuse veebibrauserile.

R1. Süsteemi WFM töötamise ajal on testimise käigus piiranguteta lubatud süsteemi WFM taaskäivitamine.

Eelnevalt toodud sündmuste jada ei kirjelda kahtlemata tervet süsteemi WFM funktsionaalsust – mudelipõhiste testide jaoks piiritleti tegevused, mida kasutajad süsteemis enim teevad.

Antud nõuded modelleeriti raamvara NModel abil, et süsteemi korrektset käitumist kontrollida. Mudeli ja adapteri ehitamist ning testide läbiviimist kirjeldab peatükk 4.

4. Süsteemi WFM mudelipõhine testimine

Antud peatükk kirjeldab süsteemil WFM läbi viidud mudelipõhiseid teste. Peatükk toob välja, millistest sammudest töö koosnes, millised komponendid loodi ning kuidas teste jooksutati.

4.1. Ülevaade süsteemi WFM mudelipõhistest testidest

Süsteemi WFM mudelipõhine testimine jagati kaheks etapiks.

- Esimeses etapis modelleeriti, kuidas üks kasutaja (asutuse operaator) logib süsteemi WFM sisse, käivitab ühe või mitme töötaja positsioneerimise ning saab ekraanile isiku(te) asukoha(d). Süsteemi sisemist käitumist selles etapis ei vaadeldud. Eesmärgiks oli kontrollida, kas süsteem saadab veebipäringutele adekvaatsed vastused. Kuna tegemist on süsteemi enimkasutatava funktsionaalsusega, siis oli see mudelipõhiste testide juurde asudes juba muude meetoditega – käsitsi testid, vahendiga JMeter [21] loodud testid – läbi testitud ning tõenäosus mudelipõhise testimise abil siit vigu leida oli suhteliselt väike. Siiski sobis selline stsenaarium hästi mudelipõhisesse testimisse sisseelamiseks ning aitab järgnevas mudelipõhise testimise põhimõtteid selgitada.

- Teises etapis lisati eelnevalt modelleeritud funktsionaalsusele kontroll, kas päringud jõuavad arveldussüsteemi ja logisüsteemi. Nende testide koostamine eeldas süsteemi WFM lähtekoodi muutmist, nii et WFM pöörduks arveldussüsteemi ja logisüsteemi asemel mudelipõhise testivahendi poole. Teises etapis loodud mudelid ja vastavad testiadapteri moodulid on tehniliselt tunduvalt keerukamad, kuid just huvipakkuvamad, kuna uurivad käitumist, kus võib esineda üks ebasoovitavamaid vigu, kus arveldussüsteemis võetakse süsteemi WFM lõppkasutaja arvelt raha maha, kuid teenuse tulemus – näiteks positsioneerimise teel leitud töötaja asukoht – süsteemi vea tõttu kasutajani ei jõua. Samuti kontrolliti teises etapis süsteemi käitumist taaskäivitamise ajal ning järel.

Kummaski etapis loodi süsteemi käitumist kirjeldav mudel ning adapter, mis seab mudelis määratletud abstraktsetele toimingutele vastavusse konkreetsed HTTP-päringud ja nende vastused.

Mudelipõhiste testide väljatöötamine ja läbiviimine raamvara NModel abil koosneb järgmistest sammudest:

- testitava funktsionaalsuse valimine ja nõuete spetsifikatsiooni täpsustamine;
- süsteemi testitavat käitumist kirjeldava mudeli ehitamine keeles C#, kasutades raamvara NModel pakutavaid atribuute ja andmetüüpe;
- mudeli kontrollimine raamvarasse NModel kuuluvate vahendite MPV ja MC abil;
- adapteri ehitamine, kasutades põhiliselt programmeerimiskeelt C#, kuid vajadusel kasutades ka muid vahendeid testitava süsteemi ja testimisvahendi ühendamiseks;
- vajadusel testlugude eelgenereerimine raamvarasse NModel kuuluva vahendi OTG abil;
- testide käivitamine raamvarasse NModel kuuluva vahendi CT abil.

Antud töös valiti testitav funktsionaalsus ja spetsifitseeriti nõuded peatükis 3. Käesolev peatükk kirjeldab mudeli loomist, kontrollimist, adapteri ehitamist ja testide käivitamist. Testide tulemused tuuakse järgmises peatükis.

Süsteemis WFM ei paku huvi üksnes lihtsad funktsionaalsed testid, vaid ka süsteemi käitumine koormuse all. Raamvara NModel ei ole esmaselt küll koormustestide jaoks ette nähtud, kuid koormuse tekitamine sellega on siiski võimalik.

4.2. Esimene etapp – veebipõhised testid

4.2.1. Testitav funktsionaalsus

Süsteemi WFM veebipõhise testimise all vaadeldakse olukorda, kus lõppkasutaja on asendatud testimissüsteemiga: viimane saadab süsteemile WFM veebipäringuid ning loeb süsteemist WFM tulevaid vastuseid.

Süsteemi WFM veebipõhistesse testidesse olid kaasatud kõik testitavale funktsionaalsusele esitatud nõuetes loetletud tegevused, mida lõppkasutaja saab läbi viia. Kõik need toimingud on kirjeldatud nii mudelis kui ka adapteris.

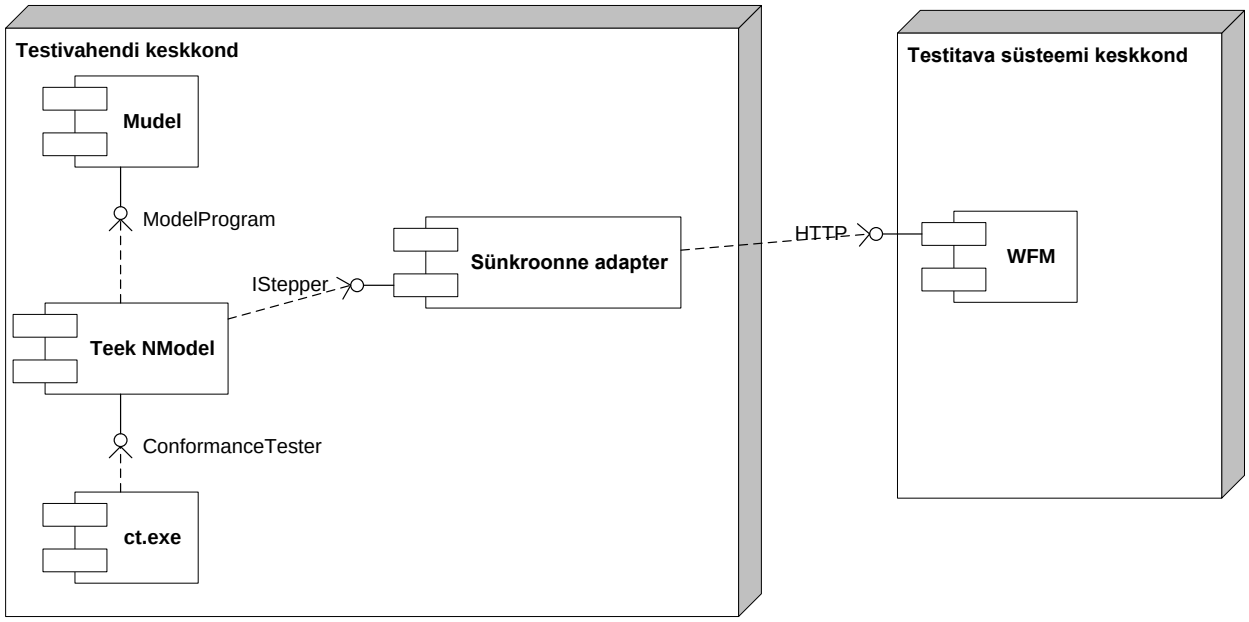
Tabel 3 loetleb funktsionaalsuse kirjeldamiseks mudelis defineeritud toimingud.

Tegevus	**Toiming mudelis**
Kasutaja sisselogimine	`WebLogin_Start, WebLogin_Finish`
Positsioneerimise algatamine	`PlacePositioningRequestSelectUser, PlacePositioningRequest`
Positsioneerimise vastuse olemasolu kontroll	`ChangedRequest_Start, ChangedRequest_Finish`
Positsioneerimise vastuse kuvamine	`GetLastKnownLocation_Start, GetLastKnownLocation_Finish`
Väljalogimine	`WebLogoff`

Tabel 3. Mudelis defineeritud toimingud.

4.2.2. Testimises osalevad komponendid

Joonis 10 annab ülevaate veebipõhises testimises osalevatest komponentidest.



Joonis 10. Süsteemi WFM veebipõhises testimises osalevad komponendid.

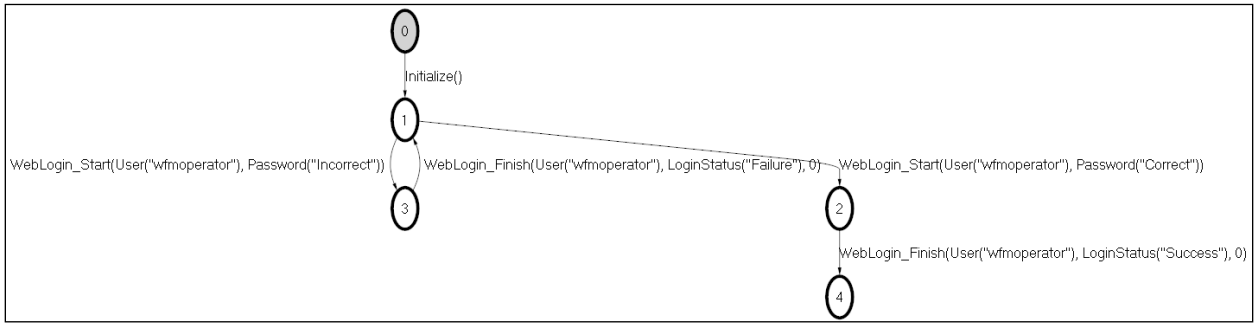
Veebipõhiste testide läbiviimiseks kirjutati järgmised komponendid:

- mudel, mis kirjeldab abstraktsel kujul süsteemi oodatava käitumise;
- sünkroonne adapter, mis seob mudelis kirjeldatud abstraktsed üleminekud süsteemile WFM vastavate konkreetsete HTTP-päringute ning nende vastustega.

Teste jooksutati raamvaras NModel sisalduva tööriistaga Conformance Tester. Nimetatud vahend käivitab testid, jälgib testide jooksmist ning kuvab kasutajale tulemused.

4.2.3. Sisselogimist kirjeldav mudel

Joonis 11 kujutab süsteemi WFM sisselogimist kirjeldavat mudelit. Joonis on sellevõrra lihtsustatud, et kirjeldab ainult ühe kasutaja ühekordset sisselogimist, näidates ära süsteemi käitumise nii vale kui õige salasõna korral. Mudel programmeeriti keeles C# ning alltoodud visuaalne kuju saadi raamvara NModel vahendi MPV abil. Olekud on skeemil tähistatud numbritega. Iga olek tähistab mingi konkreetse väärtuse vastavust igale olekumuutujale.



Joonis 11. Sisselogimist kirjeldava mudeli olekugraaf.

Antud mudel kirjeldab standardset süsteemi käitumist sisselogimisel - kasutaja võib sisestada õige või vale salasõna. Süsteem käitub vastavalt sellele:

- õige salasõna korral lubab kasutajal süsteemi sisse, andes talle vajalikud õigused (joonisel olekute jada 1 -> 2 -> 4);
- vale salasõna puhul kuvab uuesti sisselogimislehe (joonisel olekute jada 1 -> 3 -> 1).

Vahendi MPV joonistatud mudelis on igal olekul number. Iga olek tähendab erinevat mudelis defineeritud olekumuutujate väärtuste kombinatsiooni.

Tabel 4 kirjeldab olekumuutujate väärtusi antud lihtsas mudelis.

Olekumuutuja	**Tähendus**	**Muutuja väärtus olekus**				
		0	**1**	**2**	**3**	**4**
state	Süsteemi olek – kas süsteem töötab või mitte	Init	Running	Running	Running	Running
activeLoginRequests	Kasutaja-salasõna-paaride hulk parasjagu sisse logivate kasutajate talletamiseks. St kasutajad, kes on alustanud sisselogimist, kuid süsteem pole veel vastanud, kas kasutaja sisselogimine õnnestus või mitte	-	-	+ loginStatus = Success	+ loginStatus = Failure	-
usersLoggedIn	Sisse loginud kasutajate hulk	-	-	-	-	+

Tabel 4. Olekumuutujate väärtused mudelis.

Eelneva tabeli loomisel kasutatud notatsiooni kirjeldab tabel 5.

Tähis tabelis	**Tähendus**
Running, initializing	Muutuja väärtus
-	Muutujal on algne väärtus
+	Muutujal on korrektne väärtus olemas, kuid antud tabeli kontekstis ei ole oluline täpne väärtus välja kirjutada

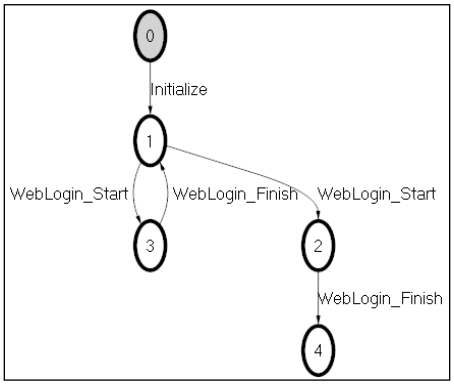
Tähis tabelis	Tähendus
+ loginStatus = Success	Muutujal on väärtus olemas ning antud tabeli kontekstis on see oluline

Tabel 5. Olekumuutujate väärtuste kirjeldamisel kasutatud notatsioon.

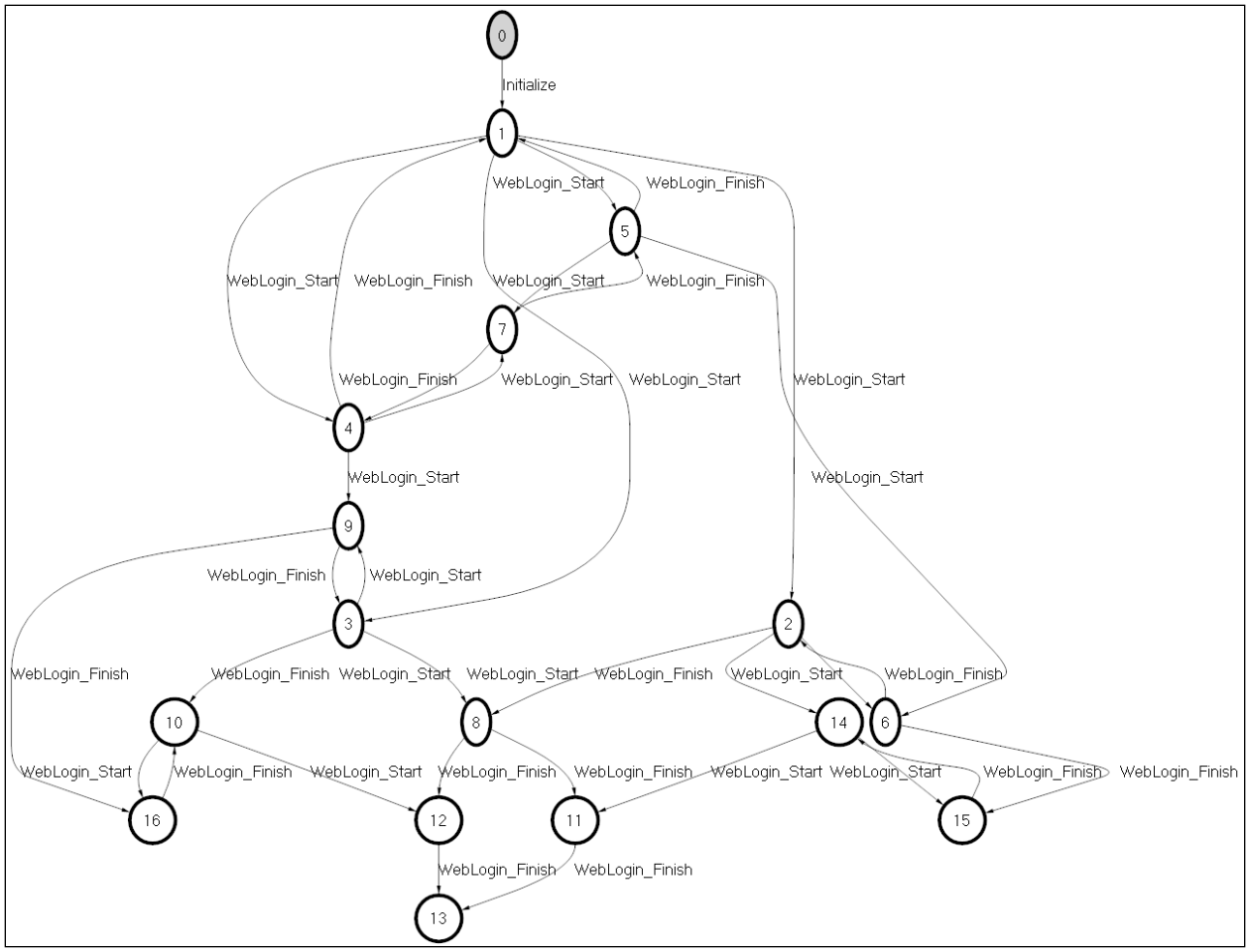
4.2.4. Mudeli keerukuse suurendamine

Eelnevalt vaadeldi väga lihtsat näidet. Süsteeme kirjeldavad mudelid on aga tavaliselt tunduvalt keerukamad ning seega ka üpris raskesti hallatavad. Joonis 12 ja joonis 13 illustreerivad, kuidas suureneb mudeli keerukus, kui mudelisse lisatakse ühe sisselogimist läbi viiva kasutaja asemele kaks.

Kummaltki pildilt eemaldati loetavuse huvides detailsed üleminekuid kirjeldavad lipikud.



Joonis 12. Sisselogimist kirjeldava ühe kasutajaga mudeli olekugraaf.



Joonis 13. Sisselogimist kirjeldava kahe kasutajaga mudeli olekugraaf.

On näha, et olekuid on kahe kasutajaga mudelis viie asemel 17. Sealjuures kirjeldati praegu ainult ühte lihtsat tegevust – sisselogimist. Lisades kasutajaid ning tegevusi (nt positsioneerimine, kusjuures positsioneerida saab ühte või mitut numbrit korraga) suureneb olekute arv veel oluliselt, muutes mudeli olekugraafi raskesti loetavaks ja mudeli raskesti hallatavaks. Tekib nn olekuplahvatus, millega toime tulemiseks võimaldab raamvara NModel mudeli üksusteks jagada. Üksustest on juttu järgmises alampeatükis.

4.2.5. Üksused

Raamvara NModel näeb ette vahendid kompositsionaalseks modelleerimiseks. See omadus võimaldab mudeli jagada üksusteks (*feature*), mida saab ükshaaval vastavalt testieesmärgile testimudelisse kombineerida. Sisse ja välja saab üksusi lülitada nii mudeli olekugraafi visuaalsel vaatlemisel vahendiga MPV kui ka testide genereerimisel vahendiga OTG ning testide

käivitamisel vahendiga CT.

Üksused võib raamvaras NModel jagada iseloomu poolest kaheks.

- Ühest küljest on üksused vahend, mis võimaldab mudelit alamosadeks jagada. Eraldi üksustena saab defineerida näiteks sisselogimise, positsioneerimise käivitamise ja väljalogimise. Nii on võimalik erinevaid süsteemi osi testida: näiteks testida kõigepealt, kas sisselogimine töötab (lülitades sisse ainult sisselogimist puudutava üksuse), seejärel lisades näiteks positsioneerimise (lülitades lisaks sisselogimist puudutavale üksusele sisse ka positsioneerimist käsitleva üksuse) ja nii edasi.

- Teisest küljest on üksused vahend teatud kitsenduste realiseerimiseks. Neid kitsendusi saab samuti ükshaaval sisse ja välja lülitada. Näiteks võib eraldi üksusena realiseerida piirangu, et sisse logida püütakse ainult õige kasutajanime ja salasõnaga ning vale salasõna ei proovita. Testide genereerimisel ja käivitamisel, samuti mudeli vaatamisel vahendiga MPV on võimalik valida, kas antud piirangut kasutada või mitte.

Joonis 14 kujutab koodinäidist, mis illustreerib kitsendavate üksuste kirjeldamist raamvaras NModel. Näites on tegemist üksusega, mis loob piirangu, et testimisvahend testib süsteemi sisse logimist ainult õige kasutajanime ja salasõnaga. Antud üksus defineerib üle toimingu WebLogin_Start ning sellele vastava üleminekutingimuse. Nii toimingu kui selle üleminekutingimuse "standardjuht" defineeriti mudeli selles osas, kus on kirjeldatud sisselogimine. Antud üksuses seatakse toimingu üleminekutingimusele täiendav piirang: sisselogimise toimingut võib alustada ainult siis, kui sisendiks anti õige salasõna. Piirangu sisselülitamine on vajalik, kui konkreetsete testide läbiviimisel ei paku huvi katsed vale salasõnaga sisse logida. Mitmed süsteemid blokeerivad kasutajakonto, kui sama kasutaja proovis mitu korda järjest vale salasõnaga sisse logida, kuid seda funktsionaalsust käesoleva süsteemi nõuetes ei kirjeldatud.

```
/// <summary> ...
[Feature("CorrectPassword")]
public static class CorrectPassword {

    /// <summary> ...
    [Action]
    public static void WebLogin_Start(User user, Password password) { }

    /// <summary> ...
    public static bool WebLogin_StartEnabled(User user, Password password) {
        return (password == Password.Correct);
    }

}
```

Joonis 14. Kitsendust seadev üksus "CorrectPassword".

Tabel 6 näitab, millisteks üksusteks jagati süsteemi WFM veebipõhistes testides loodud mudel.

Üksus	**Selgitus**
Login	Kirjeldab süsteemi sisse logimise
LogOff	Kirjeldab süsteemist välja logimise
PositioningRequest	Kirjeldab positsioneerimise käivitamise ja positsioneerimistulemuse küsimise

Tabel 6. Süsteemi WFM mudeli alamosad.

Tabel 7 näitab, millised kitsendusi realiseerivad üksused mudelisse lisati.

Üksus	**Selgitus**
OneLogin	Seab piirangu, et kui kasutaja on süsteemi juba sisse loginud, siis teda uuesti sisse logida ei püüta. Et uuesti sisse logida, peab kasutaja enne välja logima
CorrectPassword	Seab piirangu, et kasutajat püütakse süsteemi sisse logida ainult õige kasutajanimega

Tabel 7. Mudelisse lisatud kitsendavad üksused.

4.2.6. Testide käivitamine

Süsteemi WFM veebipõhistes testides kasutati käigupealt testimist: testimisvahend läbis korraga nii mudelit kui testitavat süsteemi ning kontrollis nende üksteisele vastavust. Mudeli jagamine üksusteks võimaldas teste inkrementaalselt ehitada – kõigepealt sai mudelis realiseerida ning adapteri abil testitava süsteemiga ühendada sisselogimise, siis väljalogimise ning seejärel positsioneerimise.

Testid käivitati vahendiga CT, millele anti ette konkreetseid teste kirjeldavad argumendid. Argumendid võib vahendile CT ette anda otse käsurealt, kuid mugavam on ette valmistada argumendifail.

Järgnevalt tuuakse lihtne näidis vahendi CT argumendifailist.

```
# ctargs_login.txt
# sisaldab vahendi CT argumente sisselogimise testimiseks
/r:WFM.dll
/r:Adapter.exe
/iut:WFMAdapter.WFMWFM44a.Create
WFM.Contract.CreateLoginModel
```

Antud argumendifail sisaldab järgmisi viiteid.

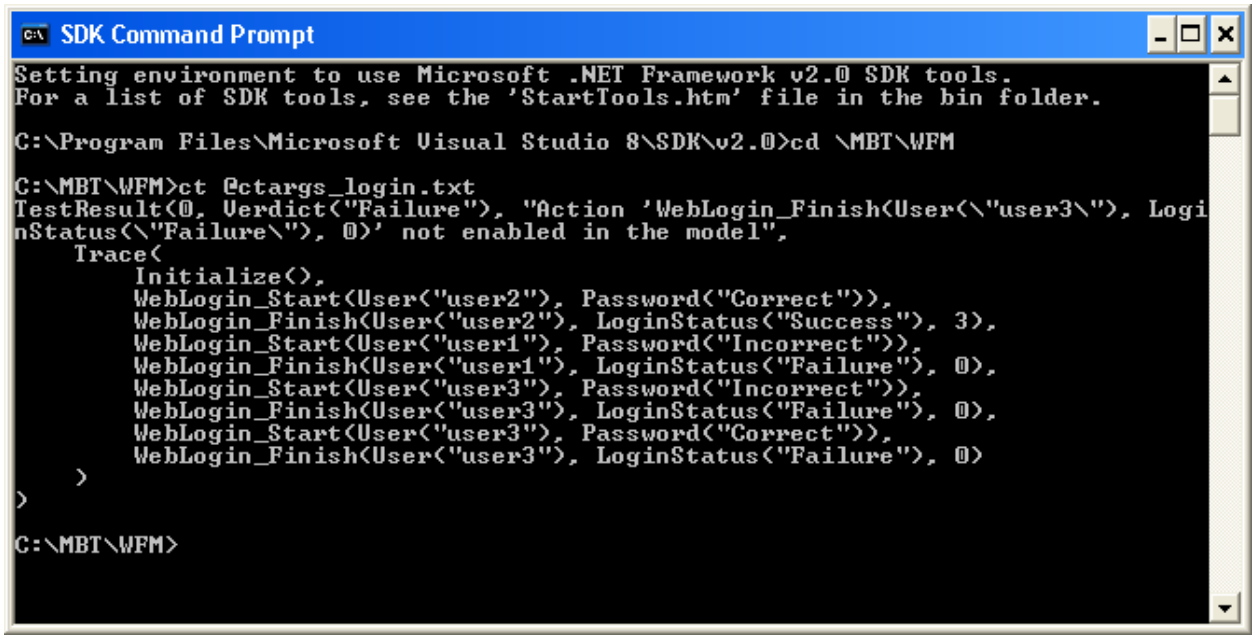
- WFM.dll ja Adapter.exe on vastavalt mudel ja adapter, mis on mõlemad .NET binaarfailid. Tähis "/r" tuleneb sõnast *reference*.
- WFMAdapter.WFMWFM44a.Create on viide meetodile adapteris, mis loob konkreetsel aadressil paikneva testitava süsteemi adapteri objekti. Samas adapteris võivad olla defineeritud ühendused mitmesse testitavasse süsteemi, näiteks kui testitavast süsteemist on käigus mitu eksemplari. Tähis "/iut" tuleneb sõnadest *implementation under test*.
- WFM.Contract.CreateLoginModel on mudeli konstruktor, kus defineeritakse see osa mudelist, mida antud testides tuleb läbida.

Testide käivitamine toimub käsurealt käsuga

```
ct @ctargs_login.txt
```

Vaikimisi kuvatakse testide läbimist ja tulemusi puudutav informatsioon käsuaknas. Joonis 15 kujutab olukorda, kus käivitatakse sisselogimise test ning see jõuab olukorrani, kus õige

salasõnaga sisselogimine ei õnnestu.



Joonis 15. Mudelipõhise testi käivitamine.

Joonis 15 esitab vahendi CT väljundis toiminguid, mis läbi viidi. Kõigepealt süsteem initsialiseeriti ning seejärel logiti süsteemi sisse kasutaja *user2*. Salasõna oli õige ning logimine õnnestus (Password("Correct") -> LoginStatus("Success")). Järgmisena prooviti vale salasõnaga sisse logida kasutajat *user1*. Probleem tekkis, kui kasutajat *user3* prooviti pärast vale salasõnaga katset sisse logida õige salasõnaga, kuid sisselogimine ebaõnnestus. Testimisvahend sai testitavast süsteemist signaali, et õige salasõnaga sisselogimise tulemusena tagastati kasutajale uuesti login-aken koos veateatega, et salasõna on vale. Sellist käitumist aga mudel ei luba ning seetõttu test ebaõnnestuski.

Eelpool toodud kasutajate identifikaatorid *user1*, *user2*, *user3* on abstraktsed, mudelis väärtustikuna defineeritud kasutajad. Adapter seab nendele vastavusse reaalsed süsteemi WFM kasutajad. Samuti on adapteris defineeritud kasutajate salasõnad.

4.3. Teine etapp – arveldus- ja logisüsteemi liidese testid

4.3.1. Testitav funktsionaalsus

Teenusepakkuja vaatevinklist üks ebameeldivamaid olukordi süsteemi WFM käitumises on see, kui kliendilt võetakse teenuse eest küll raha, kuid süsteemi vea tõttu kasutajale teenust ei osutata.

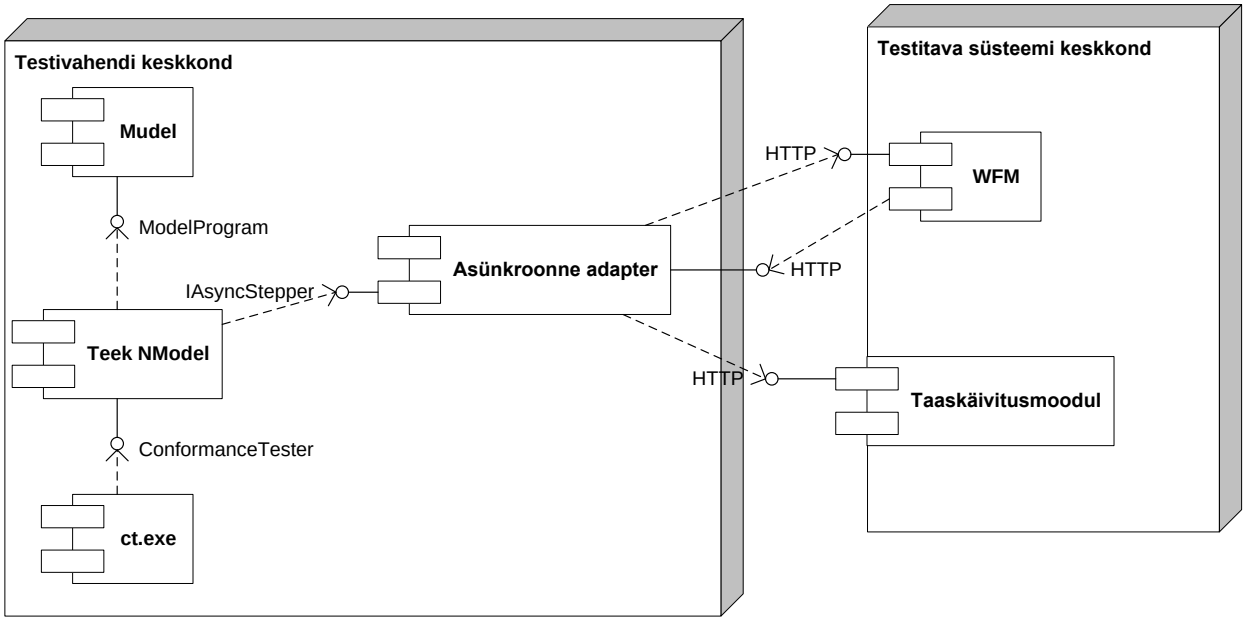
Suurima tõenäosusega on sellise olukorra põhjuseks ootamatu katkestus mõne serveri töös. Sellise olukorra uurimisele keskenduti, lisades eelpoolkirjeldatud veebipõhistele testidele arveldus- ja logisüsteemi liidese testid.

Töö käigus vaadeldi, kui palju esineb juhtumeid, kus arveldussüsteemis võetakse kasutajalt raha maha, kuid pärast seda logisse midagi ei kirjutata. Sel juhul eeldati, et kasutaja ei saanud ka teenust, mille eest temalt raha võeti.

Lisaks nimetatud juhtumite kontrollimisele lisati siin etapis testidesse süsteemi WFM taaskäivitamine. Sellega sooviti kontrollida, mis juhtub nende positsioneerimispäringutega, mis on taaskäivitamise hetkeks käivitatud, kuid mis lõppkasutajale veel vastust ei ole andnud.

4.3.2. Testimises osalevad komponendid

Joonis 16 annab ülevaate arveldus- ja logisüsteemi testides osalevatest komponentidest.



Joonis 16. Arveldus- ja logisüsteemi testides osalevad komponendid.

Arveldus- ja logisüsteemi testide läbiviimiseks kirjutati järgmised komponendid või täiendati veebipõhistes testides osalenud vastavaid komponente.

- Mudel – veebipõhiste testide jaoks kirjutatud mudelit täiendati uute, arveldus- ja logisüsteemi ning süsteemi taaskäivitamist puudutavate toimingutega.

- Adapter – muudeti selles etapis asünkroonseks, nii et saaks jälgida testitavast süsteemist asünkroonselt saabuvaid toiminguid. Adapterisse lisati veebiserver, millele süsteemi WFM lisatud adapterikomponent saadab päringu järgmistel juhtudel:

    1. süsteemis WFM toimus pöördumine arveldussüsteemi poole;
    2. süsteemis WFM toimus pöördumine logisüsteemi poole;
    3. süsteem WFM on pärast taaskäivitamist taas töös.

    Veebiserver võimaldab asünkroonsel adapteril süsteemist WFM tulevaid päringuid registreerida. Rakendati keeles C# kirjutatud avatud lähtekoodiga veebiserverit [32].

- Taaskäivitusmoodul – loodi lihtne Java Servlet, mida vastavalt veebiaadressilt välja kutsudes on võimalik teha süsteemile WFM taaskäivitus. Taaskäivitusmoodul sisaldab ka vahendit kontrollimaks, millal süsteemi WFM taaskäivitus lõppeb. Kui süsteemi WFM pärast taaskäivitust taastub, saadab moodul päringu adapteris asuvasse veebiserverisse ning teavitab nõnda testimisvahendit süsteemi WFM taas töötamisest. Taaskäivitusmoodul ei ole küll osa testitavast süsteemist WFM, kuid paigaldati füüsiliselt samasse serverisse, kus jookseb SUT. Seetõttu kujutab joonis 16 taaskäivitusmoodulit testimisvahendi keskkonnast eraldi.

- Modifikatsioonid testitavas süsteemis WFM – süsteemis WFM tehti muudatus, et lisaks standardsele arveldus- ning logisüsteemile pöörduks WFM testimisvahendi poole – alati, kui WFM kasutajalt raha maha võtab või vastava teate logisse kirjutab, annab WFM sellest ka testimisvahendile teada, saates päringu kokkulepitud veebiaadressile. Suhtlus süsteemi WFM ja testimisvahendi vahel toimub ka sel suunal üle HTTP. Üle HTTP toimuv suhtlus sai siin valitud, kuna see oli kõige lihtsam realiseerida. Samas oleks olnud võimalik HTTP-suhtluse asemel kasutada ka alternatiivseid lahendusi.

Teste jooksutati, nagu esimeseski etapis, raamvaras NModel sisalduva tööriistaga Conformance Tester.

4.3.3. Jälgitavad toimingud

Süsteemi WFM arveldus- ja logisüsteemi testid erinevad veebipõhistest testidest lisaks suurenenud keerukusele ka asünkroonsete toimingute esinemise poolest. Teisisõnu sisaldab antud testides kasutatav mudel *jälgitavaid toiminguid* ehk toiminguid, mida ei saa testivahendiga esile kutsuda, vaid mille toimumist testitavas süsteemis saab testivahend ainult jälgida ja registreerida.

Jälgitavad toimingud tulevad kasutusele seoses vajadusega kontrollida, kas kõik positsioneerimised jõuavad edukalt nii arveldus- kui ka logisüsteemi. Neid toiminguid ei saa testimisvahendiga otseselt esile kutsuda, kuna neid ei kutsu ka lõppkasutaja otseselt välja, vaid päringu saatmine arveldus- ja logisüsteemi toimub sündmuste ahelas teiste sündmuste loogilise jätkuna. Antud testides pakub huvi olukord, kus päringute saatmine arveldus- ja logisüsteemi on *positsioneerimise* käivitamise loogiline jätk. Positsioneerimise käivitab süsteemi WFM kasutaja kasutajaliideses, kuid päringud arveldus- ja logisüsteemi saadab edasi süsteem WFM.

Kuna süsteemi WFM mudelipõhistesse testidesse haarati ka süsteemi taaskäivitamise funktsionaalsus, on oluline, et testivahendisse jõuaks info selle kohta, millal süsteemi WFM taaskäivitus lõppeb ja WFM taas normaalselt töötab.

Jälgitavad toimingud realiseeriti antud töös järgmiselt.

- Mudelis kirjeldati jälgitavate toimingutega seotud üleminekud ning üleminekutingimused. Jälgitavateks toiminguteks on:
    1. DoBilling – arveldussüsteemi poole pöördumine;
    2. LogHistory – logisüsteemi poole pöördumine;
    3. RestartDone – teade sellest, et taaskäivitus edukalt lõppes ning süsteem WFM taas töötab.
- Mudelit täiendati nii, et see peaks arvet kolmes faasis positsioneerimiste üle. Nendeks faasideks on:
    1. algatatud positsioneerimised, mis ei ole veel arveldussüsteemi jõudnud;

2. positsioneerimised, mis on jõudnud arveldussüsteemi, aga mitte logisüsteemi;

3. positsioneerimised, mis on jõudnud arveldussüsteemi ja logisüsteemi (neid loetakse lõpetatud positsioneerimisteks).

- Mudelis defineeriti lubatud lõppolekutingimus (*AcceptingStateCondition*) – töö on korrektne lõpetada siis, kui kõik positsioneerimised on lõpetatud.

- Uute, jälgitavate toimingute valguses lisati mudelisse kaks uut üksust. Neid esitab tabel 8.

Üksus	**Selgitus**
BillingAndHistory	Kontroll, kas positsioneerimised jõuavad korrektselt arveldus- ja logisüsteemi.
Restart	Süsteemi taaskäivitamine juhuslikul hetkel koos kontrolliga, millal taaskäivitus edukalt lõppeb.

Tabel 8. Jälgitavate toimingutega seotud üksused.

Seega saab testidesse eraldi lülitada nii arveldus- ja logisüsteemi poole pöördumise kontrolli funktsionaalsust kui ka taaskäivitamist. Antud üksusi on võimalik teiste, töö esimeses etapis defineeritud üksustega kombineerides testida näiteks olukorda, kus ainsaks funktsionaalsuseks on mitme kasutajaga sisse logimine ning samal ajal võib toimuda süsteemi WFM taaskäivitus.

4.3.4. Mittedeterminismi näide süsteemi WFM testimisel

Süsteemi WFM funktsionaalsuses, mida antud töös testitakse, sisaldub olukord, kus ühele sisendsõnumile vastab mitu lubatud väljundsõnumit ning testimisvahend peab oskama väljundsõnumist sõltuvalt edasi käituda.

Nimelt on testitavas funktsionaalsuses järgmised punktid.

…

16. Veebibrauser küsib ettenähtud ajavahemiku tagant veebiserverilt, kas veebiserveris on uusi asutuse töötajate asukohaandmeid. Seda teeb veebibrauser alati, kui kasutaja on töötajate nimekirja lehel, hoolimata sellest, kas antud kasutaja algatas antud sessiooni

käigus positsioneerimise või mitte. Seda selletõttu, et sama asutuse töötajate positsioneerimise võib käivitada ka mõni teine kasutaja ja sama asutuse mistahes kasutaja käivitatud positsioneerimise tulemused ilmuvad siin aknas nähtavale.

17. Veebiserver tagastab veebibrauserile loenduri väärtuse: 0 – uusi positsioneerimistulemusi ei ole laekunud; 1 – uued positsioneerimistulemused on olemas. Kui uusi tulemusi pole, liigub protsess tagasi sammu 16. Kui uued tulemused on olemas, liigub protsess sammu 18.

18. Veebibrauser küsib üle HTTP töötajate viimased positsioneerimistulemused.

…

Mudelipõhiste testide töö ajal edastab adapter süsteemist WFM loetud loenduri väärtuse mudelile ning mudelis on üleminekutingimuste ning vastavate muutujate väärtustamise abil kirjeldatud lubatud tegevused loenduri ühe või teise väärtuse korral.

Lisaks esineb mittedeterminism süsteemist WFM testiadapterile laekuvate jälgitavate sõnumite järjekorras. Arveldus- või logisüsteemi poole pöördumine võib toimuda suvalisel hetkel ning testimisvahend peab oskama sellele reageerida.

4.3.5. Testide käivitamine

Teste, milles on kasutusel jälgitavad toimingud, käivitatakse vahendiga CT – seda analoogiliselt testidega, kus jälgitavaid toiminguid ei ole. Vahendile CT tuleb siin juurde anda mõned parameetrid.

Järgnevalt tuuakse näidis vahendi CT argumendifailist, mis sisaldab ka jälgitavaid toiminguid.

```
# ctargs_full.txt
# sisaldab vahendi CT argumente positsioneerimise, arveldus- ja logisüsteemi pöördumise testimiseks.

/r:WFM.dll
/r:Adapter.exe
/r:WebServerAdapter.exe
/r:HttpServer.dll
/iut:WFMAdapter.WFMWFM44a.Create

# Jälgitavad toimingud
/o:DoBilling
```

```
/o:LogHistory

# Toimingud, mis on lubatud pärast seda, kui parameetris steps
määratud sammude arv on läbi, kuid mudelis ei ole jõutud lubatud
lõppolekusse. Nende toimingute läbimise eesmärk on jõuda lubatud
lõppolekusse.
/c:ChangedRequest_Start
/c:ChangedRequest_Finish
/c:GetLastKnownLocation_Start
/c:GetLastKnownLocation_Finish
/c:placePositioningRequest
/c:placePositioningRequestSelectUser

# Sammude arv ühes testistsenaariumis
/steps:50

# Testi käivituste arv
/runs:100

WFM.Contract.CreateFullModel
```

Koondmudelprogrammi, mida vahend CT eelnevalt toodud näidises kasutab, haaratakse järgmised üksused:

- "Login" – testidesse haaratakse sisselogimise funktsionaalsus;
- "OneLogin" – kitsendus, mis lubab kasutajal sisse logida ainult siis, kui ta ei ole veel sisse logitud;
- "CorrectPassword" – kitsendus, mis lülitab välja katsed logida sisse vale salasõnaga;
- "PositioningRequest" – testidesse haaratakse positsioneerimise käivitamise ja vastuse lugemise funktsionaalsus;
- "BillingAndHistory" – testidesse haaratakse kontroll positsioneerimissündmuste jõudmisese üle arveldus- ja logisüsteemi.

Et just need üksused testi haaratakse, on kirjeldatud mudeli konstruktoris WFM.Contract.CreateFullModel.

Testide tulemuse raporteerib raamvaras NModel sisalduv vahend CT järgmiselt. Tegemist on

niisiis ühe stsenaariumiga, mis genereeriti ja käivitati süsteemi WFM mudelipõhisel käigupealt testimisel, ja mis andis positiivse tulemuse – test läbiti ilma vigu leidmata.

```
TestResult(0, Verdict("Success"), "",
    Trace(
        Initialize(),
        WebLogin_Start(User("user2"), Password("Correct")),
        WebLogin_Finish(User("user2"), LoginStatus("Success"), 1),
        placePositioningRequestSelectUser(User("user2")),
        WebLogin_Start(User("user1"), Password("Correct")),
        WebLogin_Finish(User("user1"), LoginStatus("Success"), 3),
        placePositioningRequest(User("user2"), Set<string>("0")),
        DoBilling("0"),
        LogHistory("0"),
        placePositioningRequestSelectUser(User("user2")),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), true),
        GetLastKnownLocation_Start(User("user2")),
        GetLastKnownLocation_Finish(User("user2")),
        placePositioningRequest(User("user2"), Set<string>("0")),
        DoBilling("0"),
        LogHistory("0"),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false),
        ChangedRequest_Start(User("user2")),
```

```
        ChangedRequest_Finish(User("user2"), false),
        placePositioningRequestSelectUser(User("user2")),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false),
        ChangedRequest_Start(User("user2")),
        ChangedRequest_Finish(User("user2"), false)
    )
)
```

Kui kasutada vahendi CT argumendina võtit /metrics+, on võimalik sisse lülitada statistika. Statistikat väljastab vahend CT testide kohta järgmisel kujul.

```
Test Metrics(
    General Summary(
         100 tests Executed,
         6 tests Failed,
         Pass Rate: 94.0%
    ),
    Failed Actions(
         (WebLogin_Start, Reason: The remote server returned an error:
(500) Internal Server Error.( 6 times ))
    ),
    Total time spent in each action(
         GetLastKnownLocation_Start(Executed   711   times,   Average
execution time: 00:00:00.0200000) : 00:00:14.2812500,
         placePositioningRequestSelectUser(Executed    1216    times,
Average execution time: 00:00:00) : 00:00:00.8281250,
         ChangedRequest_Start(Executed 3170 times, Average execution
time: 00:00:00.5220000) : 00:27:36.2500000,
         Initialize(Executed  100  times,  Average  execution  time:
00:00:00) : 00:00:00.0468750,
         placePositioningRequest(Executed    1141    times,    Average
execution time: 00:00:00.0710000) : 00:01:21.3593750,
         WebLogin_Start(Executed 284 times, Average execution time:
00:00:00.1270000) : 00:00:36.3437500
    )
)
```

5. Testide tulemused

Antud peatükk loetleb süsteemi WFM mudelipõhise testimise käigus saavutatud tulemused.

5.1. Veebipõhiste testide tulemused

Veebipõhiseid teste oli enne mudelipõhise testimise algust tehtud käsitsi ja vahendiga JMeter ning funktsionaalsus, mida mudelipõhiselt testiti, oli eelnevalt üpris korralikult läbi testitud. Seega võis mudelipõhise testimise juurde asudes oletada, et suure tõenäosusega veebipõhised testid uusi vigu välja ei too. Mudelipõhistes testides jõuti siiski järgmiste veaolukordadeni.

- Vahendi CT käivitamisel hakkasid koormuse all sisselogimised vahel ebaõnnestuma, nii et kasutajale anti pärast katset õige kasutajanime ja salasõnaga sisse logida süsteemne vealehekülg.
- Leiti olukord, kus pärast positsioneerimist tuli kogu aeg veebiserverist vastus, et uusi asukohaandmeid ei ole.
- Leiti viga raamvaras NModel: vahend CT käitus teatud parameetrite kombinatsiooni korral ebaõigelt. Kuna raamvara NModel on avatud lähtekoodiga, parandati see viga ära.
- Leiti viga raamvaras NModel, mis oli seotud raamistiku .Net nõrkade viidetega (*weak references*) ning raamvara NModel olekute vahemäluga. Seetõttu töötas testimisvahend Microsofti .Net-keskkonnas vigaselt, kuid Solarise operatsioonisüsteemis platvormi Mono kasutades korrektselt. Viga ilmnes toimingu parameetri väärtuste edastamisel adapterist mudelisse.

Eelnevas leiab kinnitust, et testimine on mitmesuunaline protsess. Kui mõni test ei õnnestu, siis võib vigane olla nii testitav süsteem kui ka testimisvahend, selle aluseks olev platvorm või spetsifikatsioon.

Leitud vead raporteeriti ning lasti parandada. Raamvaras NModel leitud vead on raamistiku NModel praeguses veebist kättesaadavas versioonis parandatud.

5.2. Arveldus- ja logisüsteemi liidese testide tulemused

Arveldus- ja logisüsteemi liidese testid ei leidnud normaaltingimustel süsteemi töös uusi vigu:

algatatud positsioneerimispäringud jõuavad nii arveldus- kui ka logisüsteemi. Põhjuseks, miks mudelipõhised testid siin vigu ei leidnud, on kindlasti suuresti selles, et süsteem WFM oli mudelipõhiste testide juurde asumisel küllaltki küps ning antud funktsionaalsus oli läbi testitud.

Süsteemi taaskäivitamisel leidis kinnitust, et positsioneerimispäringud, mis taaskäivitamise hetkel olid „pooleli“, läksid kaotsi. Pärast seda, kui süsteemi WFM töö oli taastunud, ei tulnud positsioneerimispäringutele vastust. Siin on aga oluline asjaolu, et reaalses töökeskkonnas asuvad süsteemi WFM serverid klastris ning kui ühe serveri töö mingil põhjusel halvatakse, võtab tööjärje üle teine server. Antud magistritöös kasutatud süsteemi WFM konfiguratsioon oli lihtsustatud, klastreid ei kasutatud.

5.3. Mudelipõhisele testimisele kulutatud ressurss

Tabel 9 annab hinnangulise ülevaate töömahtude jaotumisest mudelipõhise testimisvahendi ehitamisel. Testimisvahendi ehitamisele kulunud aeg kokku oli ca 3 inimkuud.

Testivahendi osa või tehtud töö	**Koodiridade arv**	**Kulutatud aeg (%)**
Mudel	1000	40
Asünkroonne adapter ilma veebiserverita	850	38
Veebiserveri osa adapteris	200	15
Taaskäivitusmoodul	125	4
Muudatused süsteemi WFM koodis	125	3
Kokku	**2300**	**100**

Tabel 9. Mudelipõhisele testimisele kulutatud ressurss.

Nii kulutatud aja hinnang kui ka koodiridade arv on indikatiivsed. Koodiridade arvu juures on arvesse võetud ka tühje ridu ning kommentaariridu. Kulutatud aeg ei ole lineaarses sõltuvuses koodiridade arvust, kuna erinevate komponentide loomine oli erineva keerukusega, samuti kasutati mitmeid erinevaid programmeerimis- ja skriptikeeli (C#, Java, kesta skriptid), millest mõnega oli töö autoril olnud rohkem, mõnega vähem kokkupuudet. Samuti oli mõne töö puhul – näiteks süsteemi WFM koodi modifitseerimine – kasutada antud süsteemiosa hästi tundva spetsialisti abi, mis kiirendas oluliselt tööd.

Tabelist on näha, et küllalt suur osa – 40% ajast – kulus mudeli ehitamisele. Ülejäänud tegevused, millele kulus hinnanguliselt 60% ajast, olid seotud mudeli ja testitava süsteemi

omavahelise ühendamisega ehk need võib koondada ühisnimetuse *adapter* alla.

5.4. Hinnang mudelipõhise testimise tulemuslikkusele

Antud töös vaadeldud testide läbiviimine kujunes huvitavaks, ent küllalt palju takistusi sisaldanud ja ajamahukaks ettevõtmiseks.

Positiivsete tulemustena süsteemi WFM vaatenurgast vaadatuna on olulisemad järgmised.

1. Mudelipõhine testimine aitas süsteemis WFM leida vigu, mida varem ei olnud leitud.
2. Mudelipõhine testimine jättis automaatsete testide läbiviimiseks vabamad käed kui senikasutatud automaattestimise vahendid (JMeter).

Peamiseks negatiivseks asjaoluks osutus mudeli ning adapteri loomise ning käimapaneku keerukus ning sellest tulenevalt pikk õppimisaeg.

See funktsionaalsus, mida antud töös raamvara NModel abil testiti, moodustas küllalt väikese osa süsteemi WFM kogufunktsionaalsusest. Süsteemi WFM funktsionaalsuse mudelipõhine testimine üksnes kasutajaliidese kaudu, loobudes testimast teisi süsteemi WFM liideseid, oleks senise tööga võrreldes kordades suurem töö. Samuti, kuna süsteemi WFM tuleb iga kliendi jaoks mõnevõrra kohendada – vastavalt kliendi soovile kujundust muuta, mõned funktsionaalsused välja lülitada, mõned lisada, mõned muuta, siis tuleks mudelipõhise testimise korral iga kliendi jaoks vastavalt ka mudelit ja adapterit muuta.

Kokkuvõte

Antud töös vaadeldi veebirakenduse WFM mudelipõhist testimist raamvara NModel abil. Esmalt anti ülevaade mudelipõhise testimise teooriast. Seejärel vaadeldi mitmeid saadaolevaid mudelipõhise testimise vahendeid. Nende seast valiti käesolevas töös kasutamiseks tasuta raamvara NModel. Nimetatud raamvara abil ehitati testimisvahend, mille abil testiti süsteemi WFM veebiliidest ning kahte liidest väliste tarkvarakomponentidega suhtlemiseks.

Tegemist on teadaolevalt esimese detailselt dokumenteeritud veebirakenduse testimise näitega, kus rakendatakse raamvara NModel. Testimissüsteem toetab asünkroonseid sõnumeid ja oskab süsteemist laekuva sõnumi põhjal otsustada, milline mudelis lubatud üleminekutest sooritati. Vaadeldud näide sisaldab nii jälgitavatest asünkroonsetest sõnumitest tulenevat mittedeterminismi kui ka kirjeldatud mittedeterminismi, kus ühele sisendsõnumile vastab mitu erinevat väljundsõnumit ja süsteem oskab vastavalt väljundsõnumile edasi käituda.

Töö tulemused on järgmised.

1. Tehti läbi kogu protsess alates süsteemi WFM ühe alamsüsteemi spetsifitseerimisest kuni selle mudelipõhise testimiseni. Testimise käigus leiti vigu, mida varem ei olnud avastatud.

2. Identifitseeriti mudelipõhise testimise kitsaskohad rakenduse WFM laadsete süsteemide testimisel, kui testimisel kasutatakse raamvara NModel.

   - Õppimisaeg on pikk, kuna kogu protsess kätkeb endas paljusid aspekte – mudelipõhise testimise ideoloogia, raamvara NModel ja keele C# piirangud teatud konstruktsioonide realiseerimisel, piiratud võimalused raamvara NModel abil loodud testimisvahendis vigu lokaliseerida, vajadus teada väga täpselt paljusid detaile testitava süsteemi ning selle konfiguratsiooni juures, organisatoorsetest küsimustest tulenev ajakulu (sageli võib esineda olukord, kus testitavat süsteemi majutavate serverite haldusega tegeleb omaette osakond, kõige täpsem teadmine süsteemi ülesehitusest on allhankija käes jm).

   - Adapteri ehitamine on mahukas tehniline töö, mis vajab suuremat automatiseerimist.

3. Leiti paar viga raamvaras NModel, mis parandati. Parandused laeti üles raamvara NModel veebilehele.

Töö autorile oli mudelipõhise testimise rakendamisülesanne huvitav ning andis kogemusi mitmes valdkonnas.

1. Mudelipõhine meetod andis uue lähenemisviisi testimisele tervikuna.
2. Antud mudelipõhiste testide loomine eeldas programmeerimist keeltes C# (mudelipõhise testimisvahendi ehitamine) ja Java (testimisvahendi poole pöördumise realiseerimine testitavas tarkvaras), samas oli töö autori viimaste aastate programmeerimiskogemus vähene.
3. Töö autori põhitegevuseks ei ole asukohapõhise tarkvara loomine, nii andis töö läbiviimine hea ülevaate ka selles valdkonnas toimuvast.

Tööl on mitmeid edasiarendusvõimalusi.

1. Ehitatud mudelit saab vajadusel rakendada süsteemi WFM logide analüüsil, tehes mõned muudatused süsteemiadapteris.
2. Kui süsteemi WFM mudelipõhiselt testima hakati, oli süsteem sisuliselt valmis. Mudelipõhist lähenemist saab aga tarkvara arendusprotsessis rakendada tunduvalt laiemalt, alustades mudeli loomist juba analüüsietapis. Sellisel juhul aitab mudel kõigepealt valideerida süsteemi spetsifikatsiooni ning hiljem testida süsteemi ennast.
3. Ehitatud mudelit ja adapterit saab rakendada süsteemi WFM klastritestides, kus vaadeldakse, kuidas süsteem käitub, kui mõni klastris asuv server töötamast lakkab ja teine server peab töö üle võtma. Antud töös testiti süsteemi WFM sellist eksemplari, mille ükski komponent ei asunud klastris.
4. Mudelipõhist lähenemist saab rakendada süsteemi WFM või mõne muu veebirakenduse sisemiste alamkomponentide testimisel ja hinnata mudelipõhise lähenemise efektiivsust komponenditestide läbiviimisel.
5. Raamvara NModel saab edasi arendada, lisades sinna veebirakenduste adapterite genereerimise liidese.

Model-Based Testing of Web Applications by the Example of Location-Based Software

Master Thesis

Rivo Roo

Resume

Model-based testing is an automated testing method in which test cases are derived from a model that describes some functionality of the system under test. The model used for test generation is a formalized representation of the specification. Tests can be generated offline – first a tool is used to generate concrete test sequences from the model and then these test sequences can be run several times – or online – tests are generated and executed at once.

Model-based testing provides tools to easily create a large number of tests and thus enables better test coverage with lower costs than other automated testing methods.

In the current thesis, a case study is presented where model-based testing is applied on a component of a distributed web application WorkForce Management (WFM). The purpose of the web application is to allow subscribers to track the position of their employees. This is useful, for example, to improve the practice of a courier service. To find the location of employees, WFM uses mobile positioning.

In this study, the NModel toolkit was used to build the model and the test harness. NModel is a lightweight opensource toolkit where models are expressed in C#. NModel supports both online and offline testing and uses composition of models.

The case study was approached in two phases:

- In the 1st phase, a scenario was modeled where the user initiates a positioning of one or more employees and receives the locations of the employees on the screen.
- In the 2nd phase, the test harness and model were developed further so that it was possible to observe whether the necessary information concerning positioning requests were properly sent to the billing system and the history log. Also, a restart action was added to test the system behaviour during and after restart.

The test harness built during this study was non-trivial, because it was not only able to initiate actions in system under test and observe direct responses to these actions, but it also was able to register actions that were initiated by the system under test.

On one hand, the study was successful, because it revealed some new bugs in the system under test. On the other hand, significant amount of time was spent on building the test harness, making testing of such a web application with NModel quite expensive.

Kasutatud kirjandus

[1] Myers, G. J. (2004). The Art of Software Testing. Second Edition.

[2] Kull, A. (2009). Model-Based Testing of Reactive Systems. Doktoritöö. Tallinna Tehnikaülikool.

[3] Jääskeläinen, A., Katara, M., Kervinen, A., Heiskanen, H., Maunumaa, M., Pääkkönen, T. (2008). Model-Based Testing Service on the Web. Lecture Notes in Computer Science; Vol 5047, lk 38–53.

[4] Roo, R., Ernits, J. (2008). Model-Based Testing of a Web-Based Positioning Application. 20th Nordic Workshop on Programming Theory, Abstracts, lk 75–77.

[5] Ernits, J., Roo, R., Jacky, J., Veanes, M. (2009). Model-Based Testing of Web Applications Using NModel. Lecture Notes in Computer Science; Vol 5826, lk 211–216.

[6] Utting, M., Legeard, B. (2007). Practical Model Based Testing.

[7] http://en.wikipedia.org/wiki/Model-based_testing, 17.01.2010.

[8] http://www.ttcn-3.org/StandardSuite.htm, 17.01.2010.

[9] Schieferdecker, I., Grabowski, J., Vassiliou-Gioles, T., Din, G. (2008). The Test Technology TTCN-3. Formal Methods and Testing; Vol 4949, lk 292–319.

[10] Veanes, M., Campbell, C., Schulte, W., Tillmann, N. (2005). Online Testing with Model Programs. ACM SIGSOFT Software Engineering Notes; Vol 30, lk 273–282.

[11] Alfaro, L., Henzinger, T. (2001). Interface Automata. ACM SIGSOFT Software Engineering Notes; Vol 26, lk 109–120.

[12] Heckel, R., Lohmann, M. (2003). Towards Model-Driven Testing. Electronic Notes in Theoretical Computer Science; Vol 82 No. 6.

[13] Koopman, P., Plasmeijer, R., Achten, P. (2006). Model-Based Testing of Thin-Client Web Applications. Lecture Notes in Computer Science; Vol 4262, lk 115–132.

[14] Xie, Q., Memon, A. (2007). Designing and Comparing Automated Test Oracles for GUI-Based Software Applications. ACM Trans. Softw. Eng. Methodol; Vol 16.

[15] Baresi, L., Fraternali, P., Tisi, M., Morasca, S. (2005). Towards Model-Driven Testing of a Web Application Generator. ICWE, lk 75–86.

[16] http://www.elvior.ee/, 17.01.2010.

[17] Stepien, B., Peyton, L. Xiong, P. (2008). Framework Testing of Web Applications Using TTCN-3. International Journal on Software Tools for Technology Transfer; Vol 10, lk 371–381.

[18] Schieferdecker, I., Din, G., Apostolidis, D. (2005). Distributed Functional and Load Tests for Web Services. International Journal on Software Tools for Technology Transfer; Vol 7, lk 351–360.

[19] Grieskamp, W., MacDonald, D., Kicillof, N., Stobie, K., Wurden, F., Nandan, A. (2008). Model-Based Quality Assurance of Windows Protocol Documentation. Proceedings of the 2008 International Conference on Software Testing, Verification, and Validation, lk 502–506.

[20] Javed, A. Z., Strooper, P. A., Watson, G. N. (2007). Automated Generation of Test Cases Using Model-Driven Architecture. AST '07: Proceedings of the Second International Workshop on Automation of Software Test.

[21] http://jakarta.apache.org/jmeter/, 17.01.2010.

[22] http://www.conformiq.com/qtronic.php, 17.01.2010.

[23] http://www.uppaal.com/, 17.01.2010.

[24] http://research.microsoft.com/specexplorer/, 17.01.2010.

[25] Veanes, M., Campbell, C., Grieskamp, W., Schulte W., Tillmann, N., Nachmanson, L. (2008). Model-Based Testing of Object-Oriented Reactive Systems with Spec Explorer. Formal Methods and Testing, lk 39–76.

[26] http://www.codeplex.com/NModel, 17.01.2010.

[27] http://fmt.cs.utwente.nl/tools/torx/introduction.html, 17.01.2010.

[28] http://www-verimag.imag.fr/~async/TGV/index.shtml.en, 17.01.2010.

[29] http://www.gentleware.com/, 17.01.2010.

[30] Jacky, J., Veanes, M., Campbell, C., Schulte, W. (2008). Model-Based Software Testing and Analysis with C#.

[31] Veanes, M., Schulte, W. (2008). Protocol Modeling with Model Program Composition. FORTE '08: Proceedings of the 28th IFIP WG 6.1 international conference on Formal Techniques for Networked and Distributed Systems, lk 324–339.

[32] http://webserver.codeplex.com/, 17.01.2010.

Lühendid ja mõisted

Antud töös kasutatakse järgmisi mõisteid ja lühendeid.

Eestikeelne mõiste	Ingliskeelne mõiste	Lühend	Selgitus
Abstraktsed testid	Abstract tests	-	Testid, mis tuletatakse mudelist ja on samal abstraktsioonitasemel nagu mudel. Abstraktseid teste ei ole võimalik käivitada. Vt ka *käivitatavad testid*.
Adapter	Adapter	-	Tarkvaraline ühenduslüli testitava süsteemi ning selle käitumist kirjeldava mudeli vahel. Viib vastavusse mudelis kirjeldatud abstraktsed toimingud ning testitavas süsteemis toimuvad konkreetsed päringud ja tegevused.
Arveldussüsteem	Billing system	-	Mobiilioperaatori juures asuv süsteem, millega testitav süsteem WFM suhtleb, et kontrollida teenuse osutamiseks vajaliku raha olemasolu kliendi kontol ning teenuse osutamise eest kliendilt raha võtta.
Eelsalvestatud testlugu	Preset test	-	Mudelist genereeritud test, mida saab korduvalt testitava süsteemi peal käivitada. Inglise keeles kasutatud ka mõistet *Offline tests*, samuti *a priori testing*.

Eestikeelne mõiste	Ingliskeelne mõiste	Lühend	Selgitus
Juhitav toiming	Controllable Action	-	Toiming, mille toimumise üle otsustab testimisvahend ehk keskkond, milles testitav süsteem töötab.
Jälgitav toiming	Observable action	-	Toiming, mida ei saa testimisvahendiga käivitada ja mille teeb testitav süsteem muude tegevuste tulemusena. Testimisvahend saab selliste toimingute esinemist jälgida.
Koondmudel-programm	Contract model program	-	Mudelprogramm, mis käsitleb süsteemi kõiki lubatud käitumisi.
Käigupealt testimine	Online test	-	Mudeli ja testitava süsteemi kooskäivitamine vastupidiselt eelsalvestatud testlugude käivitamisele.
Käivitatavad testid	Executable tests	-	Testid, mille abstraktsioonitase on sobiv testide jooksutamiseks, kasutades vastavat testimistarkvara. Käivitatavad testid kujutavad endast täpseid testiskripte, milles kasutatud muutujate ja parameetrite väärtused on vastavuses testitava süsteemi vastavate väärtustega. Vt ka *abstraktsed testid*.

Eestikeelne mõiste	Ingliskeelne mõiste	Lühend	Selgitus
Lõplik olekumasin	Finite State Machine	FSM	Lõplik hulk üleminekuid olekute vahel koos algoleku ning lõppoleku(te)ga.
Lubatud lõppolek	Accepting State	-	Olek, milles olekumasina tööl on lubatud lõppeda.
Mobiilpositsioneerimise süsteem	Mobile Positioning System	MPS	Tarkvarakomponent, mis mobiilpositsioneerimist reaalselt läbi viib. Ka WFM kasutab positsioneerimiseks seda standardset tarkvara.
Mudel	Model	-	Testitava süsteemi oodatava käitumise abstraktne, osaline esitus.
Mudelprogramm	Model program	-	Testitava süsteemi oodatavat käitumist kirjeldav lihtsustatud tarkvaraprogramm.
Spetsifikatsioon	Specification	-	Kirjeldus, mis määratleb, mida testitav süsteem tegema peab.
Stsenaarium	Scenario	-	Üks konkreetne tee või piiratud teede hulk olekumasina läbimiseks.
Stsenaariummudel-programm	Scenario model program	-	Mudelprogramm, mis käsitleb ainult teatud hulka tegevuste jadasid programmis.

Eestikeelne mõiste	Ingliskeelne mõiste	Lühend	Selgitus
Testimisvahend	Test Tool	-	Üldnimetus mudelist ja adapterist ning vajadusel muudest komponentidest koosnevale tarkvarale, mille abil testitavat süsteemi testitakse.
Testitav süsteem	System Under Test	SUT	Süsteem, mida testitakse ja mida mudelprogramm kirjeldab. Inglise keeles kasutatakse ka mõistet *implementation under test (IUT).*
Testlugu	Test case	-	Tingimuste ja muutujate hulk, mille alusel testija saab otsustada, kas süsteemile esitatud nõue või süsteemi kasutuslugu on täidetud.
Toiming	Action	-	Mudelis ja testitavas süsteemis defineeritud üleminek olekute vahel.
Tsükkel	Livelock	-	Alamosa programmist, kus programm liigub teatud olekute vahel, kuid ei pääse antud olekute hulgast välja.
Tupik	Deadlock	-	Olek, kust ei ole võimalik ühessegi teise olekusse liikuda.
Unifitseeritud modelleerimiskeel	Unified Modeling Language	UML	Tarkvaraarenduses kasutatav standardne üldotstarbeline modelleerimiskeel.
Vastavus	Conformance	-	Kooskõla mudeli ja testitava süsteemi vahel.

Eestikeelne mõiste	Ingliskeelne mõiste	Lühend	Selgitus
Väärtustik	Enumeration	-	Abstraktne andmetüüp, mille väärtus kuulub konstantsete identifikaatorite hulka.
Valideerimine	Validation	-	Kontrollimine, kas programm käitub nii nagu plaanitud.
Üksus	Feature	-	Omavahel seotud muutujate ja meetodite grupp.
Üleminekutingimus	Enabling condition	-	Tingimus, mis näitab, kas toiming on lubatud. Toiming on lubatud, kui toimingu üleminekutingimus on täidetud.
-	Abstract State Machine Language	AsmL	Abstraktsete olekumasinate teoorial põhinev käivitatava spetsifikatsiooni keel. Eestikeelses tekstis kasutatakse lühendit AsmL. Mõistet ei tõlgita eesti keelde.
-	Conformance Tester	CT	Raamvaras NModel sisalduv vahend testide käivitamiseks.
-	Implementation Under Test	IUT	Vt System Under Test.
-	Model Checker	MC	Raamvaras NModel sisalduv vahend mudeli olekute ja üleminekute loendamiseks ning olekute saavutatavuse kontrolliks.

Eestikeelne mõiste	Ingliskeelne mõiste	Lühend	Selgitus
-	Model Program Viewer	MPV	Raamvaras NModel sisalduv vahend mudelprogrammide visualiseerimiseks.
-	Offline test	-	Vt Preset test.
-	Offline Test Generator	OTG	Raamvaras NModel sisalduv vahend ühenduseta testide genereerimiseks.
-	Testing and Test Control Notation Version 3	TTCN-3	Rahvusvaheliselt standardiseeritud programmeerimiskeel, mis on ette nähtud spetsiaalselt testide kirjutamiseks.
-	WorkForce Management	WFM	Asukohapõhine teenus, mis võimaldab asutustel logistikat parandada. Testitav süsteem antud töös.

Lisa 1. Süsteemi WFM mudeli ja adapteri lähtekood

Magistritööle on lisatud CD, millel on süsteemi WFM mudeli ja adapteri lähtekood koos dokumentatsiooniga.

Mudeli ja adapteri lähtekood asub kataloogis WFM. Dokumentatsioon eraldi failina asub CD juurkataloogis failis wfm_dokumentatsioon.chm.

Koodi käivitamiseks ja modifitseerimiseks peab arvutisse olema paigaldatud järgmine tarkvara:

1. Microsoft .NET Framework Version 2.0;
2. Microsoft Visual C# 2005, sobib ka tasuta versioon Express Edition;
3. Raamvara NModel.

Esimesed kaks vahendit saab laadida Microsofti kodulehelt, raamvara NModel on kättesaadav lehelt http://www.codeplex.com/NModel/.

Koodi kompileerimiseks tuleb Start-menüüst avada SDK Command Prompt, liikuda mudeli koodi juurkataloogi (kataloogi, kus asub fail WFM.sln) ja anda käsk *kompileeri*. See käsk kompileerib kogu koodi ning paneb tulemuse kataloogi *build*.

Liikudes vahendis SDK Command Prompt kataloogi *build* on võimalik kasutada järgmisi raamvara NModel vahendeid.

Vahend	**Mida võimaldab**	**Näidiskäivitus**
MPV	Mudeli kujutamine visuaalselt	mpv /r:WFM.dll WFM.Contract.CreateLoginModel
MC	Mudeli olekute ja üleminekute loendamine	mc /r:WFM.dll WFM.Contract.CreateLoginModel
OTG	Testlugude genereerimine	otg /r:WFM.dll WFM.Contract.CreateLoginModel > LoginTests.txt

Kõigil kolmel juhul on võimalik näidetes toodud konstruktori CreateLoginModel asemel argumendiks anda mõni teine mudelis defineeritud konstruktor. Erandiks on vahend MC, millele on mõttekas ette anda ainult konstruktoreid, mis ei puuduta jälgitavaid toiminguid sisaldavaid üksusi, kuna vastasel juhul on olekuruum lõpmatu. Konstruktorid on mudelis defineeritud faili WFM.cs regioonis „Konstruktorid“.

Ligipääs süsteemile WFM ei ole avalik ning seetõttu ei ole laiemal üldsusel võimalik teste süsteemil WFM käivitada.