TALLINNA TEHNIKAÜLIKOOL
Infotehnoloogia teaduskond
Informaatikainstituut
Teadmussüsteemide õppetool

Telekommunikatsiooni ettevõtte tootepakkumiste ärireeglite haldussüsteemi realiseerimine kasutades Drools reeglimootorit

Magistritöö

Üliõpilane: Natalie Karolin
Üliõpilaskood: 111264IABM
Juhendaja: Jaak Tepandi

Tallinn
2014

Autorideklaratsioon

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

(kuupäev) (allkiri)

Telekommunikatsiooni ettevõtte tootepakkumiste ärireeglite haldussüsteemi realiseerimine kasutades Drools reeglimootorit

Annotatsioon

Käesoleva töö peamiseks eesmärgiks on pakkuda välja lahendus konkreetse telekommunikatsiooni ettevõtte tootepakkumiste ärireeglite haldusega seotud protsesside ning süsteemide parendamiseks.

Töös otsib autor lahendusi ärireeglite kirjeldamise, haldamise ning dokumenteerimisega seotud probleemidele.

Töö olulisemad tulemused on parendatud ärireeglite haldusprotsess, ärireeglite haldussüsteemi prototüüp koos esialgse reeglibaasiga, mis on realiseeritud kasutades Drools reeglimootorit, ning ärireeglite testimise ja dokumenteerimise põhimõtted.

Lõputöö on kirjutatud eesti keeles ning sisaldab teksti 84-l leheküljel, 7 peatükki, 13 joonist ning 2 tabelit.

Implementation of Telecommunication Company Product Offerings Business Rule Management System Using Drools Rule Engine

Abstract

The main purpose of this thesis is to propose a solution to problems related to product offerings business rules management processes and systems in a certain telecommunication company.

In this work the author searches solutions to the problems related to describing, managing and documenting business rules.

The results, which were obtained during this work, are improved business rule management process, product offering business rule management system prototype based on Drools rule engine, initial rule base and principles of testing and documenting business rules.

The thesis is written in Estonian and contains 84 pages of text, 7 chapters, 13 figures and 2 tables.

Mõisted ja lühendid

Lühend/mõiste ***Inglise keeles***	**Kirjeldus**
Idee tõestus *Proof of Concept (POC)*	Lihtsustatud realisatsioon, nõuete valideerimiseks[24].
BPMN *Business Process Model and Notation*	Standard äriprotsesside mudelite graafiliseks esitamiseks [27].
Käivitatav spetsifikatsioon *Executable specification*	Automatiseeritud nõude täpne kirjeldus koos näidetega, mis on kättesaadav kõikidele meeskonna liikmetele [1, lk.21].
Prototüüp *Prototype*	Süsteemi projekteerimise, töönäitajate ja kasutuspotentsiaali hindamiseks või nõuete paremaks mõistmiseks või määramiseks sobiv mudel või esialgne teostus [20].
Reeglimootor *Rule Engine*	Reeglite käivituskeskkond [6, lk. 17].
Versioonimine	Hüpermeedium- andmebaasi komponentide muutuste ajaloo pidamise võime [21].
Ärireegel *Business Rule*	Täpne väide käitumisest, mis on mõnikord väljendatud KUI … SIIS vormis, mis peab olema täidetud, kui tingimused on rahuldatud[22].
Ärireeglite haldussüsteem *Business Rule Management System (BRMS)*	Süsteem, mille abil saab kirjeldada, käivitada, jälgida ning hallata ärireegleid eraldi rakenduse koodist. [28]

Joonised

Tabelid

Sisukord

Sissejuhatus

Igal ettevõttel on kindlasti olemas palju ärireegleid. Mõned nendest võivad olla lihtsamad, teised keerulisemad. Osad ärireeglid tagatakse töökorralduslikult ning osad on automatiseeritud. Selleks et tänapäeval konkurentsis püsida, peab ettevõte olema paindlik ning suutma kiiresti kohanduda muutuva keskkonnaga. Selle jaoks peab ettevõte suutma tuua lühikese aja jooksul turule uusi tooteid ja teenuseid, mis võivad vajada ka uute ärireeglite kehtestamist. Süsteemis kirjeldatud ärireeglid aitavad protsesse automatiseerida, tagada parema kontrolli ning vähendada inimliku eksimuse faktorit.

Teisest küljest on oluline, et süsteemis kirjeldatud ärireeglid oleksid läbipaistvad, ehk kõikidele huvitatud osapooltele kätte- ja arusaadavad ning kooskõlas reaalsete äriliste soovide ja vajadustega. Kahjuks võib reaalelus tihti märgata olukordi, kus süsteemis kirjeldatud ärireeglid on ebaselged, süsteemi koodi vahel laiali ning nende kohta puudub korralik ja ajakohane dokumentatsioon. Sellises olukorras puudub ettevõttes ühine ning kõikidele kättesaadav teave kehtivatest ärireeglitest, reeglite muutmine on keeruline, aeganõudev ning võib põhjustada palju vigu. Loomulikult pärsib see ettevõtte arengut ning konkurentsivõimet.

Käesoleva tööga soovib autor parendada konkreetse Eesti telekommunikatsiooni ettevõtte (edaspidi Ettevõte) tootepakkumiste ärireeglite haldust ja täiendada ärireeglite haldusprotsessi, võttes kasutusele Drools reeglimootori ja lisaks ka teised abivahendid. Põhilised probleemid, mida autor soovib lahendada on:

- ärireeglite kirjeldamise jäikus ja keerukus,
- ärireeglite hallatavuse keerukus ja vigade rohkus,
- ärireeglite läbipaistmatus ning ajakohase dokumentatsiooni puudus.

Käesoleva töö eesmärgid on järgnevad.

1. Luua parendatud tootepakkumiste ärireeglite haldamise protsess.
2. Luua Drools reeglimootori kasutamise prototüüp, veendumaks vahendi sobivuses Ettevõtte tootepakkumiste ärireeglite realiseerimiseks.

3. Luua Ettevõtte tootepakkumiste ärireeglite reeglibaas ning reeglite kirjeldamise põhimõtted, mida oleks võimalik tulevikus ära kasutada reaalsete reeglite kirjeldamisel.
4. Pakkuda välja lahendus ärireeglite testimise ning dokumenteerimise parendamiseks.

Töö on jaotatud seitsmeks suuremaks peatükiks. Töö esimeses peatükis tutvustab autor tänast olukorda Ettevõttes, seletab lahti ärireeglite haldussüsteemi mõiste ja selle komponendid ning kirjeldab võimalikke ärireeglite kirjeldamise viise. Teises peatükis püstitab autor süsteemile nõuded ning tutvustab üldist lahendust ja selle valiku põhjuseid. Kolmandas peatükis kirjeldab autor parendatud tootepakkumiste ärireeglite haldusprotsesse. Neljas peatükk on pühendatud realiseeritud Drools reeglimootori kasutamise prototüübile. Viiendas peatükis tuuakse välja prototüübi jaoks loodud reeglibaas, reeglite struktuur ning kirjeldamise põhimõtted. Kuuendas peatükis räägitakse reeglite testimise ning dokumenteerimise põhimõtetest. Viimases peatükis võtab autor kokku, kuidas loodud ärireeglite haldusprotsess ning Drools reeglimootori kasutamine vastab püstitatud nõuetele.

1. Taust

1.1. Praegune olukord ja probleemid

Täna on Ettevõttel olemas toimiv ja spetsiaalselt Ettevõtte jaoks tehtud tootekataloogi rakendus ning selle üheks osaks on ka reeglimootor, mille abil saab kirjeldada tootepakkumiste ning arvelduskampaaniate ärireegleid. Reegleid on võimalik hallata selle jaoks mõeldud kasutajaliidese abil ning tehtud reegleid hoitakse andmebaasi tabelites. Praeguse reeglimootori esialgne idee seisnes selles, et ilma tehniliste teadmisteta inimesed, nagu näiteks ärijuhid, protsessijuhid ja ärianalüütikud, saaksid ise ärireegleid kirjeldada ja hallata. Reaalselt kujunes aga olukord selliseks, et reeglite haldamise vastutus on siiski IT osakonna süsteemianalüütikute kätes, sellepärast et reeglid on läinud keeruliseks ja raskesti loetavaks. Puudub ka ühtne kõigile kättesaadav ärireeglite dokumentatsioon ja seetõttu ei ole ülevaadet sellest, millised ärireeglid on süsteemis realiseeritud ning millised peaksid olema tegelikud ärireeglid.

Reeglimootor on realiseeritud Oracle andmebaasis, kasutades PL/SQL'i keelt ning on suhteliselt piiratud funktsionaalsusega. Kasutada saab ainult ette defineeritud tingimusi ja tegevusi. Süsteemianalüütikud peavad ikka pöörduma arendajate poole, et mõnda uut tegevust või tingimust realiseerida.

Realiseeritud ärireeglid ei ole kaetud automaattestidega, mis teeb reeglite muutmise keeruliseks ning ohtlikuks tegevuseks. Muudatus ühes kohas võib põhjustada ootamatuid vigu teises kohas ning nõuab mahukat ületestimist. Tänase reeglimootori disain on selline, et isegi soovi korral oleks selle abil realiseeritud reegleid automaattestidega väga keeruline katta.

Järgnevalt on toodud kokkuvõtlik tabel (vt. Tabel 1) tänase lahenduse plussidest ja miinustest.

Plussid	**Miinused**
lihtsamaid muudatusi saab teha läbi kasutajaliidese	reeglid on tihti keerulised ja mitteloetavad äripoole inimese jaoks
muudatusi saab teha nii arendus-, kui ka otse toodangu keskkonnas	puudub ärireeglite ajakohane dokumentatsioon
muudatusi saab kanda automaatselt arenduskeskkonnast toodangu keskkonda	reeglimootori tingimuste ja tegevuste valik on piiratud
on olemas silumise funktsionaalsus	reeglimootor töötab ainult andmebaasis olevate

	andmete peal; ei sobi hästi, kui reeglimootorit soovib kasutada teine rakendus (näiteks Javas realiseeritud iseteenindus)
	saab kasutada ainult tootepakkumiste ja arvelduskampaaniate reeglite jaoks
	reeglid ei ole automaattestidega kaetud

Tabel 1 Tänase lahenduse plussid ja miinused

Oluline on mainida ka seda, et tänane tootekataloogi rakendus on üle kümne aasta vana ning on jõudnud oma elukaare lõppu ja vajab suurt refaktoreerimist või asendamist uue süsteemi vastu.

1.2. Ärireeglite haldussüsteem

Ärireeglite haldussüsteemiks (*ingl. k Business Rule Management System* (BRMS)) nimetatakse süsteemi, mille abil saab kirjeldada, käivitada, jälgida ning hallata ärireegleid eraldi rakenduse koodist [27].

Ärireegliks nimetatakse täpset väidet käitumisest, mis on mõnikord väljendatud „kui – siis“ vormis, mis peab olema täidetud, kui tingimused on rahuldatud [22].

Ärireeglite haldussüsteem koosneb minimaalselt järgmistest komponentidest:

- käivituskeskkond (reeglimootor) – reeglite käivitamine ning jälgimine,
- reeglite hoidla (reeglibaas) – reeglite hoidmine, struktureerimine, versioonihaldus ning auditeerimine,
- kasutaja tööriistad – vahendid reeglite kirjeldamiseks ning testimiseks.

Üldjuhul on ärireeglite haldussüsteem mõeldud kasutamiseks äripoole inimestele, kes saaksid jõustada automatiseeritud reegleid ilma IT osakonda kaasamata.

Ärireeglite haldussüsteemide kasutamise plussideks tuuakse tavaliselt välja järgmised[8].

- Äripoolel on suurem kontroll reeglite üle ning on väiksem sõltuvus IT osakonnast.
- Reeglid on kirjeldatud selliselt, et neid saab lugeda ka ilma IT hariduseta inimene.
- Äriloogika on eraldatud teisest koodist.

- Võimalus muuta reegleid toodangu keskkonnas tihedamini kui rakenduse versioonitsükkel.

Miinusteks tuuakse välja järgnevad [8].

- Ärireeglite haldussüsteemi arendamiseks ning juurutamiseks on vaja tugevat kompetentsi, mida võib olla keeruline turult leida ja mis võib olla ettevõtte jaoks kallis.
- IT osakonna tugi on siiski tihti vajalik, näiteks uute tegevuste või tingimuste loomiseks.
- Reeglite kaudne käitumine *(*ingl.k *implicit behavior)* võib olla raskesti mõistetav suurte reeglite arvu korral [23].
- Ühe reegli muudatus võib põhjustada palju ootamatuid vigu [23].

Turult saab leida nii tasulisi, kui ka tasuta ärireeglite haldussüsteeme, näiteks IBM WebSphere ILog, Oracle BusinessRules, JBoss Drools Guvnor, OpenRules ja paljud teised.

1.3. Ärireeglite kirjeldamise meetodid

Ärireegleid võib kirjeldada väga erinevalt. Sellest räägivad nii raamatu „Agile Business Rule Development“ autorkond [4, lk.117-127], kui ka näiteks Pete Bennet artiklis „Should I be using JBoss Rules, jBPM, SQL or just plain Java?“[3] . Järgnevalt kirjeldab autor täpsemalt mõnesid olulisemaid ärireeglite kirjeldamise meetodeid.

Osa reeglitest saab kirjeldada andmebaasi struktuurides või SQL päringute abil. See on kõige lihtsam viis kirjeldamiseks ning haldamiseks, kuid ei sobi kõikide ülesannete lahendamiseks. Andmebaasis on näiteks keeruline kirjeldada selliseid reegleid, mis võivad sõltuda paljudest erinevatest sisenditest või millel on keeruline otsustusloogika.

Teine võimalus, mis on ilmselt kõige paindlikum üldse, on kirjeldada reegleid otse süsteemi koodis. Siin on miinuseks see, et reeglid on jäigad ning suvalise muudatuse sisseviimine vajab rakenduse versioonivahetust. Suurtes ettevõtetes võivad versioonivahetused toimuda harva, ning siis ei õnnestu ka ärireegleid muuta kiiremini, kui versioonitsükkel seda ette näeb. Teiseks, teeb selline lähenemine võimatuks selle, et tehniliste teadmisteta inimesed saaksid vähemalt lihtsamaid reegleid ise kirjeldada.

Kolmandaks variandiks on kasutada reeglimootorit. Üks võimalus on arendada spetsiaalselt ettevõtte vajaduste jaoks mõeldud reeglimootor, kuid on olemas ka palju valmislahendusi.

Kusjuures valmisreeglimootorid võivad olla nii eraldiseisvad, kui ka üks osa suuremast ärireeglite haldussüsteemist.

Eelpool nimetatud meetodeid on võimalik ka kombineerida vastavalt konkreetsele vajadusele. Sellest mainivad ka raamatu „Agile Business Rule Development“ autorid [4, lk.117-127].

2. Üldine lahendus

Antud töös keskendub autor pakkumiste ärireeglite haldamisele, kuid üldine lahendus peaks tulevikus sobima ka teiste ärireeglite haldamiseks, nagu näiteks arvelduskampaaniate reeglid, tarne protsessi reeglid vms.

2.1. Nõuded lahendusele

Järgnevalt püstitab autor olulisemad mittefunktsionaalsed nõuded, mida ta soovib selles töös realiseerida. Nõuded on pandud kokku lähtuvalt autori senisest kogemusest, arvestades praeguse lahenduse kitsaskohtadega ning kasutades toimivaid lahendusi. Samuti on viidud läbi arutelud ärijuhtide, arhitektide ning tänaste reeglite kirjeldajatega. Püstitatud nõuded on kirjeldatud toote kvaliteedi standardi ISO 25010 järgi [25].

Soorituse tõhusus

- **NFR1:** Reeglimootori töö 50 pakkumise reeglite käivitamiseks < 0,5 sekundit.

Kasutatavus

- **NFR2:** Äripoole inimesed peavad saama vaadata süsteemis realiseeritud ärireegleid loetaval kujul.
- **NFR3:** Äripoole inimesed peavad saama teha ise lihtsamaid ärireeglite muudatusi.
- **NFR4:** Reeglid peavad olema struktureeritud.

Hooldatavus

- **NFR5:** Reeglid peavad olema taaskasutatavad.
- **NFR6:** Reeglid peavad olema versiooneeritavad. Peab olema võimalik taastada reegli eelmist versiooni ning jälgida muudatuste logi.
- **NFR7:** Reegleid peab olema võimalik käivitada silumisrežiimis - jälgida reeglite käivitamise logi ning vahetulemusi.
- **NFR8:** Ärireeglid peavad olema kaetud automaattestidega.

Töökindlus

- **NFR9:** Ühe reegli viga ei tohi katkestada tervet protsessi.

Porditavus

- **NFR10:** Ärireegleid peab olema võimalik muuta arenduskeskkonnas.
- **NFR11:** Ärireegleid peab olema võimalik muuta toodangu keskkonnas ilma versioonivahetuseta.
- **NFR12:** Arenduse ja toodangu süsteemi ärireeglid peavad olema omavahel sünkroniseeritavad.

2.2. Lahenduse kirjeldus

Lähtudes ärireeglite haldussüsteemi komponentidest (peatükk 1.2) ning eelmises peatükis püstitatud nõuetest pakub autor järgnevalt välja lahenduse tootepakkumiste ärireeglite haldussüsteemi realiseerimiseks.

Käivituskeskkond (reeglimootor)

Reeglite käivitamiseks ning paigaldamiseks kasutatakse reeglimootorit JBoss Drools Expert. Reeglimootori komponent võib olla reaalse süsteemi puhul nii üheks mooduliks Ettevõtte tellimiskeskkonnas, kui ka eraldiseisvaks rakenduseks. Reeglimootori kasutamist kirjeldatakse täpsemalt peatükis 4.

Reeglite hoidla (reeglibaas)

Järgnevalt toob autor välja reeglite hoidmise ning versioonimise põhimõtted.

- Tootepakkumised ja nendevahelised seosed asuvad tootekataloogi andmebaasis.
- *drl* ja *drt* formaadis reeglifailid asuvad failikataloogis.
- Reeglimallides (*drt* formaadis failid) kasutatavad andmed asuvad tootekataloogi andmebaasis.
- Tootepakkumiste ja reeglifailide vahelised seosed asuvad tootekataloogi andmebaasis.
- Kogu pakkumiste, reeglimallide andmete ning reeglifailide sisu versioonitakse versioonihaldus tarkvara GIT abil ning hoitakse keskses Ettevõte koodihoidlas.

Tehtud reeglibaasi kirjeldatakse täpsemalt peatükis 5.

Kasutaja tööriistad

Reeglite haldamiseks kasutatavate tööriistade põhimõtted on järgnevad.

- Tootepakkumisi hallatakse Ettevõtte tootekataloogi rakenduses.
- Drools reegleid hallatakse meelepärase IDE abil (näiteks Eclipse või IntelliJ IDEA).
- Ärireegleid testitakse ning dokumenteeritakse näidete abil kirjeldamise meetodiga *(*ingl.k. *Specification by Example)* kasutades selle jaoks Concordion, JUnit ning Mockito vahendeid (täpsemalt peatükis 6).

Tootekataloogi rakenduse realisatsioon ning tootekataloogis kirjeldatud pakkumiste spetsifikatsioon ei kuulu selle töö skoopi. Autori fookuseks on reeglimootori abil realiseeritud ärireeglid.

2.3. Ärireeglite kirjeldamise meetodi valik

Peatükis 1.3 tõi autor välja kolm erinevat meetodit ärireeglite kirjeldamiseks ning mainis ka seda, et neid meetodeid on võimalik kombineerida. Kombineeritud lähenemist on kasutatud Ettevõttes varemgi ning autor otsustas kasutada seda oma töös ka edaspidi, kuna sellise lähenemise puhul saab kõige paremini ära kasutada iga meetodi eeliseid.

Esialgseid tootepakkumisi ning nendevahelisi seoseid on võimalik lihtsalt ja mugavalt kirjeldada andmebaasis. Andmebaasis kirjeldatud reegleid on kõige kergem ja kiirem ka muuta ning see ei vaja tehnilisi teadmisi. Parema arusaamise jaoks toob autor mõned näited andmebaasis kirjeldatud reeglitest.

- Interneti kiiruse „200M/200M“ pakkumine on suunatud ainult äriklientidele.
- Televisiooni pakkumine vajab toimimiseks interneti ühenduse olemasolu.
- Täiendavate vaatamiskohtade pakkumiste maksimaalne lubatud arv on 9.
- Interneti portide parameetri lubatud väärtused on: Avatud või Suletud.

Selliseid ärireegleid, mis on kõikide pakkumiste puhul ühised, kohustuslikud ning tagavad andmete terviklikkuse, on kõige kindlam realiseerida otse koodis, selleks et need oleksid alati tagatud. Siia hulka sobivad näiteks erinevad kontrollid.

- Kui pakkumine ei ole pakutav, siis ei tohi seda ostukorvi lisada.
- Kui pakkumise parameetri väärtus ei ole lubatud, siis ei tohi seda valida.
- Kui pakkumise maksimaalne lubatud arv on ületatud, siis pakkumine ei ole valitav.

Keerulisemaid reegleid, mis sõltuvad kontekstist ning mida ei ole võimalik või on keeruline andmebaasis kirjeldada, realiseeritakse reeglimootori abil. Reeglimootor võtab oma töö aluseks andmebaasis kirjeldatud pakkumised, kuid võib nende pakkumiste valikut piirata või muuta pakkumiste esialgseid häälestusi (näiteks vaikeväärtused, lubatud väärtuste nimekirjad jne.).

Näiteks võib tuua järgmised ärireeglid, mida oleks mõistlik kirjeldada reeglimootori abil.

- Ruuteri OneAccess pakkumine on lubatud ainult juhul, kui aadressi prioriteetne tehnoloogia on SDSL või GHSDS, või Interneti kiiruse pakkumiseks on valitud „8M/8M“.
- Kui klient on pensionär, siis talle on lubatud pakkuda kuni 6 telefoni sõbranumbrit, muul juhul on lubatud sõbranumbrite arv 3.
- Erakliendile on tähtajalise kasutuse tingimustel seadmepakkumine lubatud juhul kui klient ei ole kunagi varem nimetatud tingimustel seadet sellel tootel saanud või kui uue seadme vajadus on tingitud tehnoloogia vahetusest optika või voip tehnoloogiale.

2.3.1. Drools reeglimootori valik

Reeglimootori valikul lähtus autor Ettevõtte arhitektuurilisest suunast liikuda võimaluse korral eemale kommertstarkvaradest vabavaraliste lahenduste suunas. Seetõttu langesid valikust välja sellised ärireeglite haldussüsteemid nagu näiteks IBM WebSphere ILog või Oracle BusinessRules. Vabavaralistest reeglimootoritest luges autor lähemalt reeglimootoritest OpenRules, JESS ning JBoss Drools. JBoss Drools oli nendest kõige levinum ning kõige suurema kogukonnaga reeglimootor. Lisaks sellele oli autor juba varasemalt tuttav Drools reeglimootoriga teisest ülikooli jooksul tehtud projektist. Nendel põhjustel otsustas autor realiseerida esimese prototüübi kasutades just Drools reeglimootorit. Käesoleva töö raames realiseeritava prototüübi peamine mõte on kontrollida vahendi vastavust püstitatud nõuetele. Ei ole välistatud ka töö tulemus, mis näitab, et Drools reeglimootor ei ole siiski sobiv.

JBoss Drools Expert reeglimootori puhul ei kujuta reeglid endast midagi muud, kui kokkulepitud formaadis (näiteks *drl, drt*) failide kogumit. See annab võimaluse kirjeldada reegleid otse failides

koos ülejäänud rakenduse koodiga (arendajale kõige mugavam). Soovi korral on võimalik ehitada uue funktsionaalsuse või kasutada valmislahendusi ka reeglite muutmiseks läbi kasutajaliidese, mis sobiks paremini, näiteks, ärianalüütikutele. Lisaks sellele võib tuua välja ka järgmised põhjused, mis räägivad Drools reeglimootori kasuks.

- Drools on avatud lähtekoodiga vaba tarkvara.
- Droolsi taga on suur kogukond, kelle käest saab kindlasti abi küsimuste või probleemide korral.
- Drools reeglimootor on väga paindlik, tingimuste ning tegevuste hulk on piiramatu.
- Drools sobib kasutamiseks arendajatele, kuid selle peale on võimalik ehitada ka kasutajaliidese analüütikute jaoks.
- Drools reeglimootor kasutab levinud ekspertsüsteemide põhimõtteid (faktid, RETE algoritm).
- Reeglimootor on kiire tänu optimeerimise algoritmidele.
- Drools reeglimootor on realiseeritud Java keeles, mis sobib hästi Ettevõttes soositud tehnoloogiatega.
- Drools pakub mugavamat süntaksit reeglite kirjeldamiseks kui tavaline Java kood.
- Head auditeerimise võimalused on sisse ehitatud.
- Drools on hea raamistik ärireeglite kirjeldamiseks Javas [5].

Kui üldjuhul on reeglimootori või terve ärireeglite haldussüsteemi kasutusse võtmise üheks eesmärgiks ärireeglite haldamise võimalus tehniliste teadmisteta inimestele, siis autori arvamusel, ei toimi selline lähenemine reaalelus hästi. Sellele viitab oma artiklis Fowler [23] ning seda näitab ka Ettevõtte kogemus, et lõpuks hakkasid ärireegleid haldama ikkagi süsteemianalüütikud ning arendajad, mitte äripoole inimesed, kellele see algselt mõeldud oli.

Läbi kasutajaliidese reeglite kirjeldamine piirab oluliselt reeglimootori funktsionaalsust ning uute tingimuste ning tegevuste loomine vajab pidevalt arendajate ressurssi. Lisaks sellele võib keeruliste reeglite muutmine põhjustada süsteemi töös vigu. Seega peaksid sellised muudatused niikuinii läbima ka testimise tsükli ning olema kaetud automaattestidega. Nendel põhjustel on antud töös otsustatud, et raskemaid ärireegleid kirjeldab üldjuhul arendaja, muutes reeglifaile otse samast kohast, kust ka ülejäänud süsteemi koodi. Analüütikute jaoks jääb võimalus hallata ise lihtsamaid ärireeglite muudatusi (andmebaasis hoitavate pakkumiste muutmine, tabelis hoitavate andmete

muutmine). Sellisele lähenemisele viitab ka Wiseman artiklis „Real-World Rule Engines“ [29]. Kui järgmistes projekti skoopides selgub, et kasutajaliides reeglite kirjeldamiseks on siiski vajalik, võib kaaluda JBoss poolt pakutava ärireeglite haldussüsteemi Drools Workbench kasutusse võtmist või täiesti uue Ettevõtte vajadusi rahuldava kasutajaliidese arendamist.

3. Ärireeglite haldamise protsess

Selles peatükis kirjeldab autor ärireeglite haldamisega seotud protsesse ning rolle, mis on vajalikud edukaks ärireeglite haldamiseks organisatsioonis. Iga protsessi kirjeldus koosneb protsessi diagrammist, mis on tehtud BPMN notatsioonis, ning protsessi tekstilisest kirjeldusest.

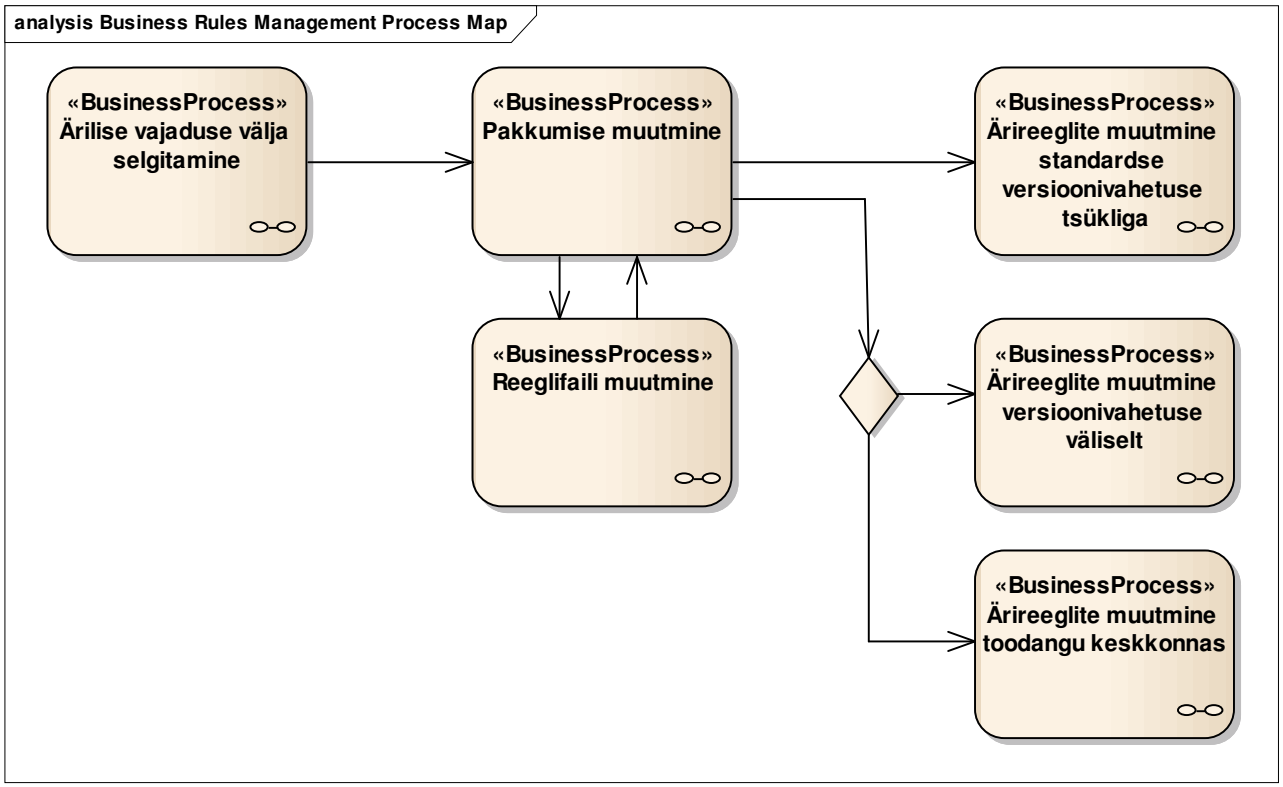
3.1. Rollid

Rollide kirjeldamiseks võtab autor aluseks raamatus „Agile Business Rules Development“ toodud rolle [4, lk. 61-64], kuid täiendab neid Ettevõtte konteksti jaoks sobivamaks. Raamatus kirjeldatud rollidest sobivad Ettevõtte konteksti järgmised 4 rolli.

- **Äritellija** *(*ingl.k. *SME - Subject Matter Expert)*
  Annab ärilise sisendi (ärinõuded) ning vaatab hiljem üle ärireeglite dokumentatsiooni, veendumaks, et süsteemis kirjeldatud reeglid on õiged. Üldjuhul täidavad seda rolli Ettevõttes ärijuhid, tootejuhid, valdkonnajuhid või protsessidisainerid.
- **Ärireeglite analüütik** *(*ingl.k *Rule Analyst)*
  Tunneb hästi äriloogikat ning samal ajal teab, kuidas toimib süsteem ning reeglimootor. Analüüsib ärinõudeid, otsib vastuolusid ning liiasusi. Mõtleb läbi terminoloogia. Analüütik on ühenduslüli äritellija ning ärireeglite kirjeldaja vahel. Võib ise teha väiksemaid ärireeglite muudatusi, näiteks tootepakkumistes või otsustustabelites.
- **Ärireeglite kirjeldaja** *(*ingl.k. *Rule writer)*
  Arendaja kompetentsiga inimene, kes realiseerib kokkulepitud reegleid ning kirjutab ka automaatteste. Peab olema varakult kaasatud. Kirjeldaja ning analüütiku rolli võib tihti täita sama inimene.
- **Ärireeglite arhitekt** *(*ingl.k. *Rule architect)*
  Laiendab tavalise arhitekti rolli. Lisaks tavalise arhitekti ülesannetele mõtleb läbi reeglibaasi struktuuri, tegeleb reeglimootori jõudluse tagamisega, mõtleb läbi reeglibaasi sõltuvused ning domeenimudeli kasutuse. Töötab välja üldised reeglite kirjeldamise põhimõtted ning mallid. Teeb järelvalvet, et süsteemis kirjeldatud reeglid oleksid kooskõlas põhimõtetega ning kokkulepetega.

Kuigi igal rollil on omad ülesanded ning vastutusala, toimib protsess hästi ainult siis, kui nende rollide vahel toimub pidev kommunikatsioon ning koostöö.

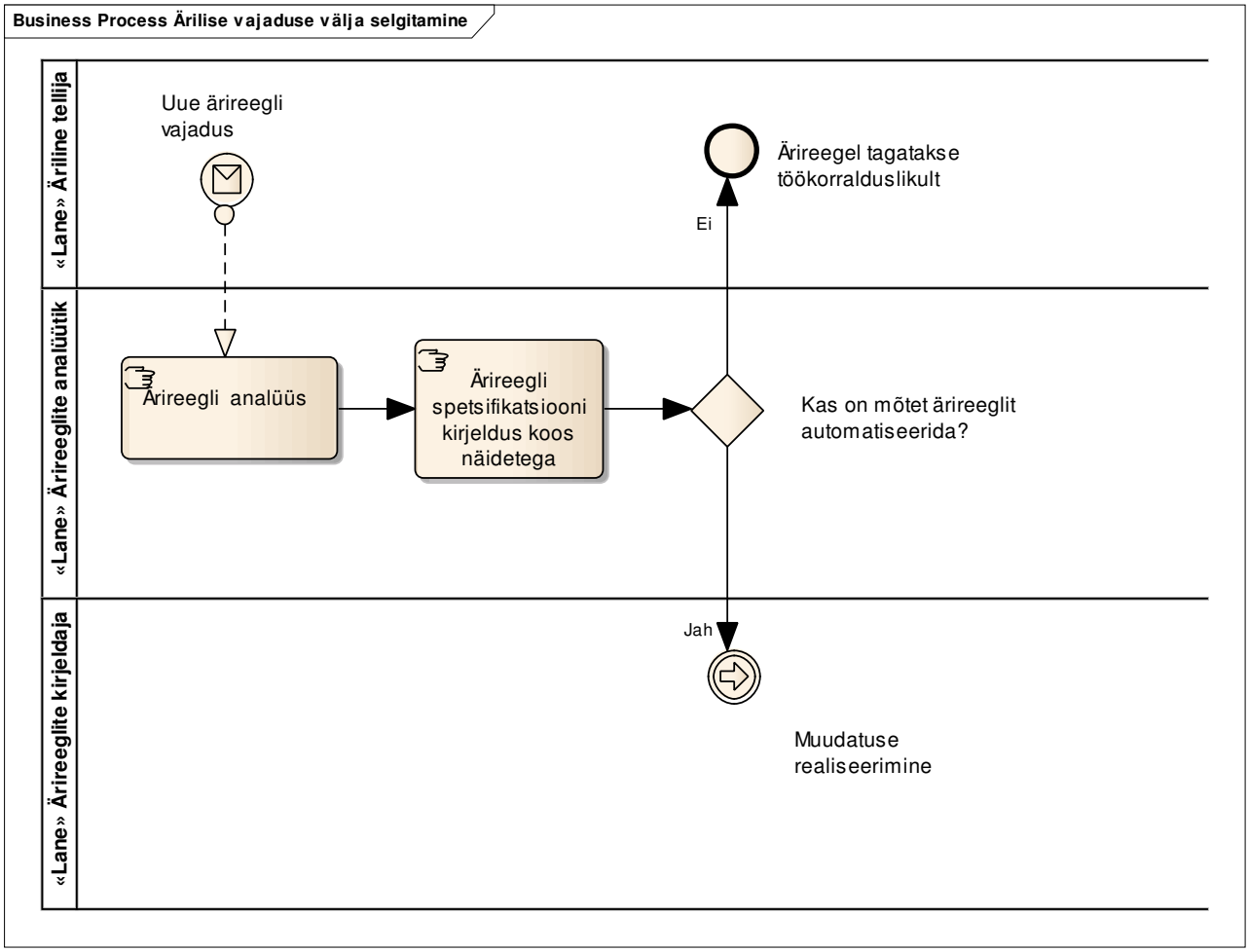
3.2. Protsessikaart



Joonis 1 Protsessikaardi diagramm

Protsessikaardil (vt. Joonis 1) on toodud välja põhilised tootepakkumiste ärireeglite haldusega seotud protsessid. Ärireeglite haldamise vajadus tekib sellel põhjusel, et äriliselt on vaja kehtestada uut ärireeglit. Ennekõike tuleb selgitada välja täpne äriline vajadus. Kui ülesanne on selge, siis tuleb täiendada või lisada vastavad pakkumised ja vajadusel ka reeglifailid. Edasi läheb üldjuhul protsess standardse versioonivahetuse tsükliga, kus muudatus realiseeritakse arenduskeskkonnas ning versioonivahetusega kantakse toodangu keskkonda. Vajadusel on võimalik muuta ärireegleid versioonivahetuse väliselt või otse toodangu keskkonnas.

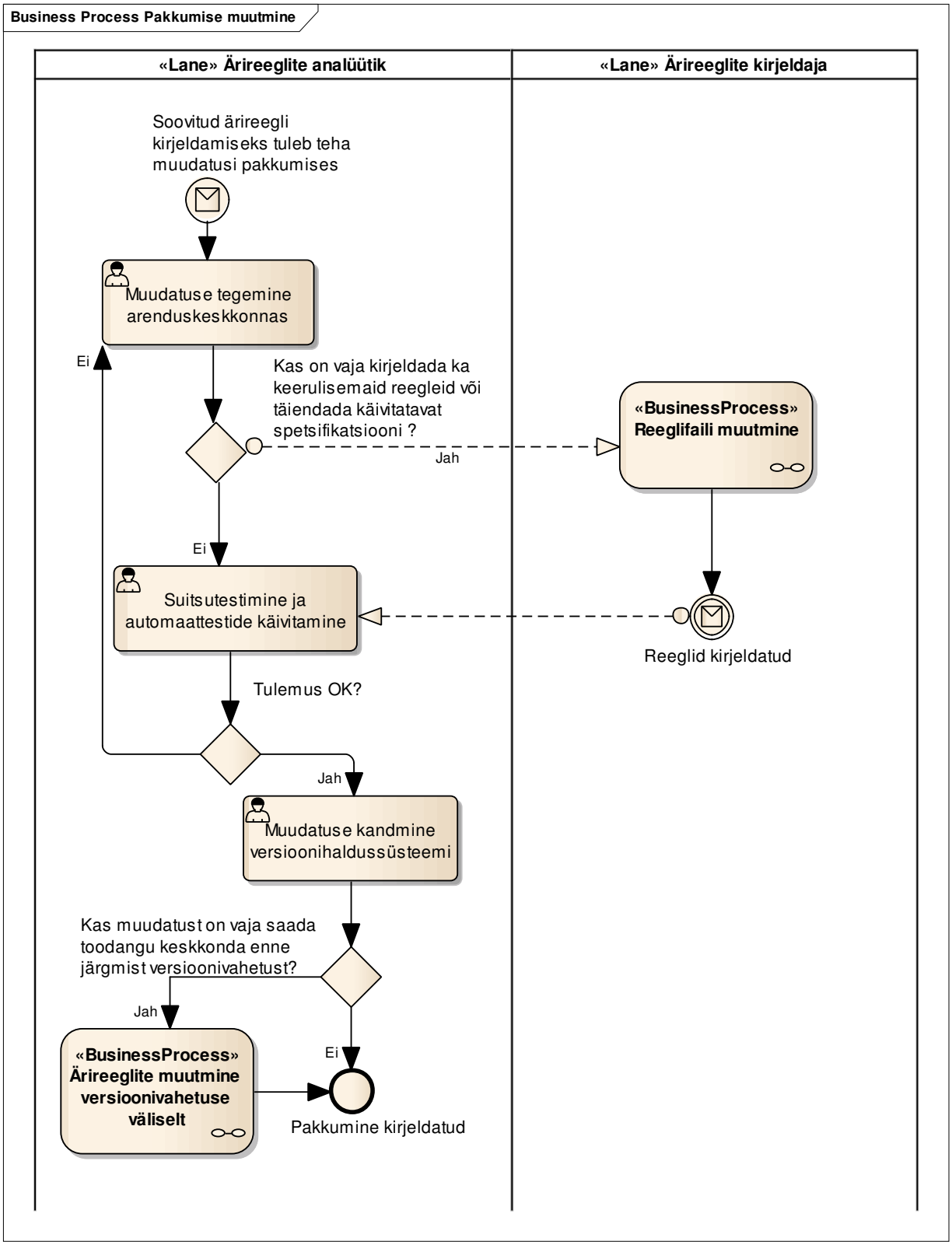
3.3. Ärilise vajaduse välja selgitamine



Joonis 2 Ärilise vajaduse väljaselgitamise protsessi diagramm

Kui äritellijalt tuleb sisendiks uue ärireegli vajadus, siis ärireeglite analüütik peab seda eelkõige analüüsima ning täpsustama. Näiteks küsima täiendavaid küsimusi, otsima vastuolusid või kitsaskohti. Kui esmane analüüs on tehtud, siis pannakse kirja täpne ärireegli spetsifikatsioon koos detailsete näidetega ärireegli kohta („Specification by Example“ meetod on täpsemalt kirjeldatud selle töö kuuendas peatükis). Seda sammu on soovitatav teha koos terve meeskonnaga (äriline tellija, analüütik, kirjeldaja, testijad jt), selleks et kõikidel osapooltel tekiks ühene arusaam sellest, mida on vaja teha. Kui ärireegli kirjeldus on paigas, siis mõnikord võib selguda, et sellisel kujul ei olegi seda võimalik süsteemis realiseerida (kas on liiga kallis või näiteks süsteemis puudub vajalik info otsuste tegemiseks). Sellisel juhul, kui ärireeglit süsteemis ei realiseerita, lepitakse kokku töökorraldus, kuidas ärireegel tagada, näiteks teenindajate koolitamine, teadete/hoiatuste lisamine, aruandlus ja järelevalve vms. (vt. Joonis 2)

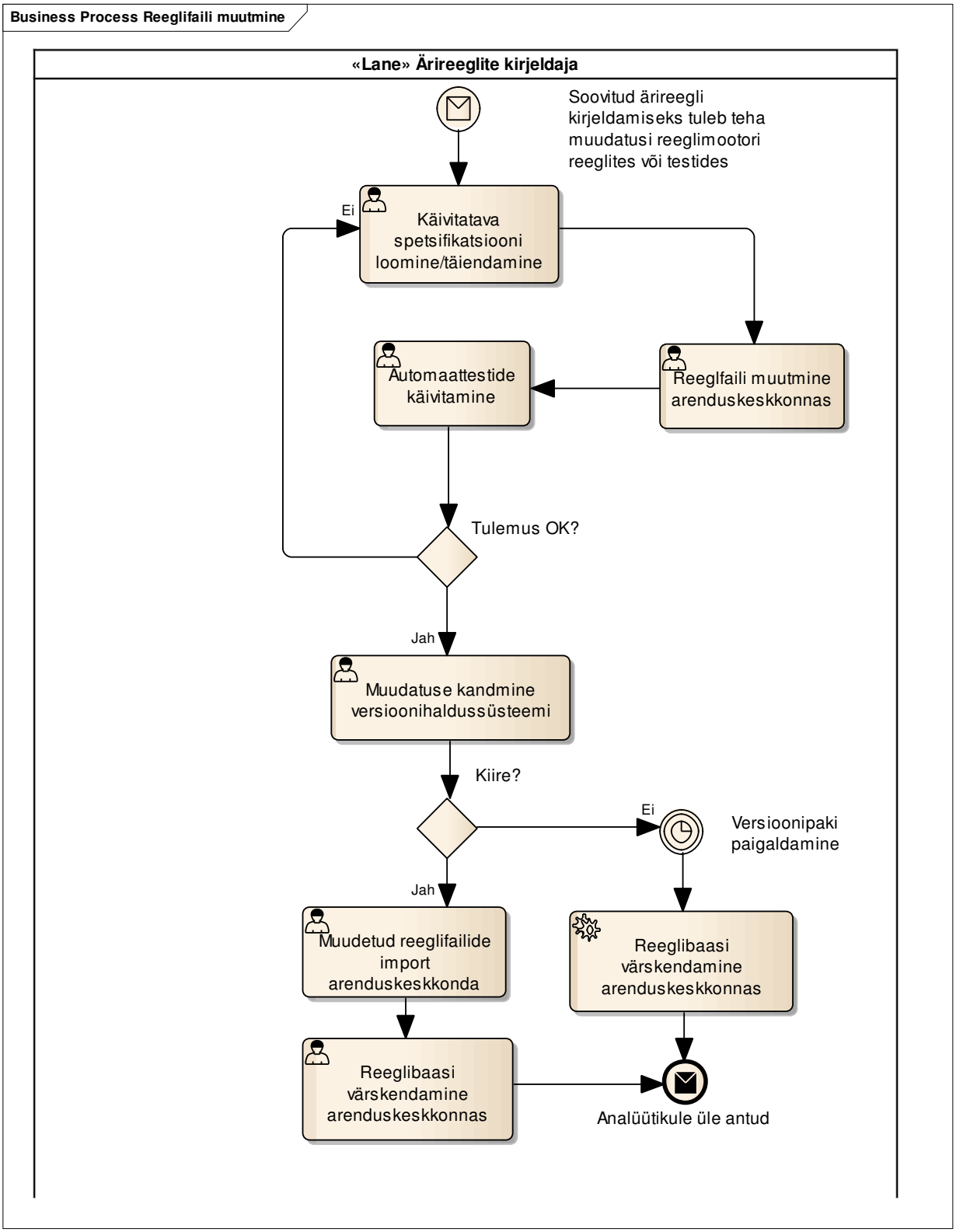
3.4. Pakkumise muutmine



Joonis 3 Pakkumise muutmise protsessi diagramm

Pakkumise muutmisega (uue pakkumise lisamine, olemasoleva pakkumise muutmine) saab ärireeglite analüütik üldjuhul ise hakkama. Kui pakkumise kirjeldamisel ei õnnestu analüütikul soovitud reeglit kirjeldada, siis peab ta pöörduma ärireeglite kirjeldaja poole, kes saab vajaliku reegli realiseerida reeglimootori vahendite abil (tulevikus saab teha ka ärireeglite analüütiku jaoks lihtsama kasutajaliidese reeglifailide muutmiseks). Samuti võib analüütik pöörduda ärireeglite kirjeldaja poole ka siis, kui on vaja täiendada või kirjutada mõni uus käivitatav spetsifikatsioon. Kui pakkumise kirjeldus on valmis, peab ärireeglite analüütik veenduma, et see töötab. Selle jaoks käivitab ta arenduskeskkonnas automaatteste ning vajadusel teeb ka suitsutesti läbi kasutajaliidese. Kui esineb vigu, siis pöördub analüütik tagasi kirjeldamise sammu juurde ning protsess kordub, kuni suitsutest õnnestub. Kui kõik on korras, siis peab analüütik kandma oma muudatused versioonihaldussüsteemi, kust järgmisel versioonivahetusel võetakse pakkumise seis ning paigaldatakse see toodangu keskkonda. Kui pakkumine peab jõudma toodangu keskkonda enne versioonivahetust, siis tuleb selle jaoks teha paar lisategevust, mis on kirjeldatud protsessis 3.7 „Ärireeglite muutmine versioonivahetuse väliselt“. (vt. Joonis 3)

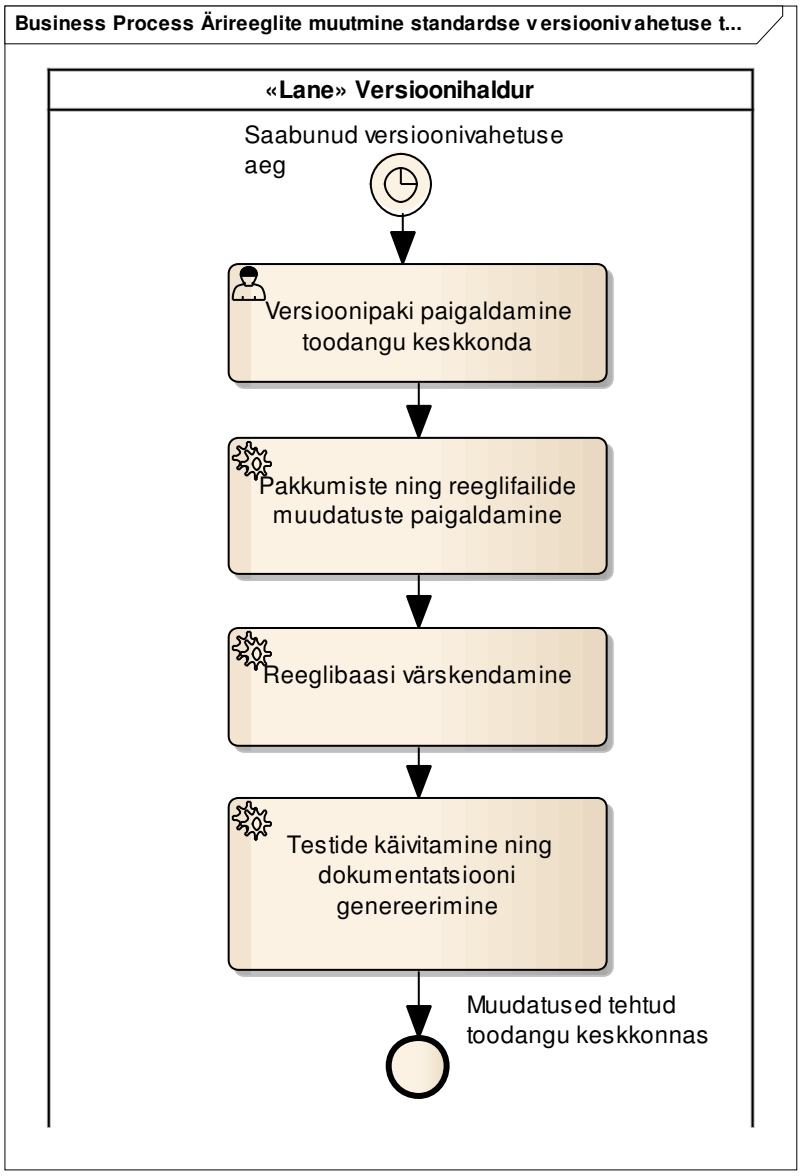
3.5. Reeglifaili muutmine



Joonis 4 Reeglifaili muutmise protsessi diagramm

Kui ainult pakkumiste häälestustest ei ole piisav ärireegli tagamiseks, siis vajab ärireeglite analüütik ärireeglite kirjeldaja abi. Koos täiendatakse käivitatavat spetsifikatsiooni, pärast mida muudab kirjeldaja lokaalselt reeglifaile ning kontrollib reeglite korrektsust käivitades automaatteste. Kui testide tulemused on positiivsed, siis kannab kirjeldaja oma muudatused versioonihaldussüsteemi. Kui muudatusega on kiire ning on soov hakata kohe arenduskeskkonnas suitsuteste tegema, siis peab kirjeldaja käsitsi importima muudetud reeglifaile arenduskeskkonda ning käivitama reeglibaasi värskendamise. Kui kiiret ei ole, siis arenduskeskkonnas järgmise versioonipaki paigaldamisel värskendab süsteem automaatselt ka reeglibaasi. (vt. Joonis 4)

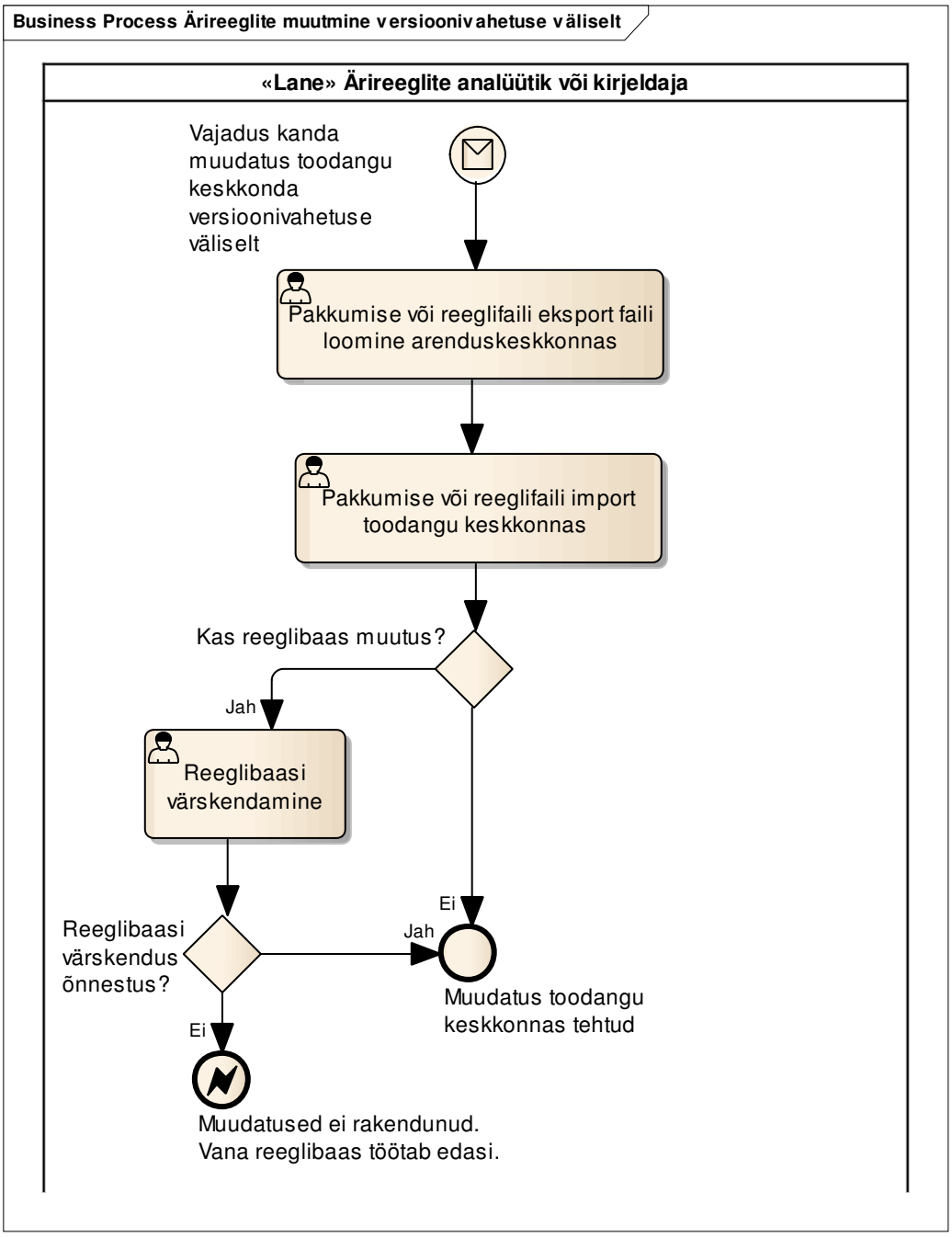
3.6. Ärireeglite muutmine standardse versioonivahetuse tsükliga



Joonis 5 Ärireeglite muutmine standardse versioonivahetuse tsükliga

Standardse protsessi järgi on reeglite muutmine toodangu keskkonnas üks osa suuremast versioonivahetuse protsessist. Versioonipaki paigaldamisel kantakse pakkumiste ning reeglifailide muudatused automaatselt üle toodangu keskkonda. Ühe sammuna toimub ka automaatne reeglibaasi värskendamine ning seejärel ka testide käivitamine, mille tulemusena toimub dokumentatsiooni värskendamine. Selle protsessi juures eeldatakse, et versioonipaki paigaldamine on testkeskkonnas läbi testitud ning reeglibaasi värskendamisel ei tohi vigu tekkida. Vigade tekkimisel toimub standardne eelmise versiooni taastamise protsess. (vt. Joonis 5)

3.7. Ärireeglite muutmine versioonivahetuse väliselt

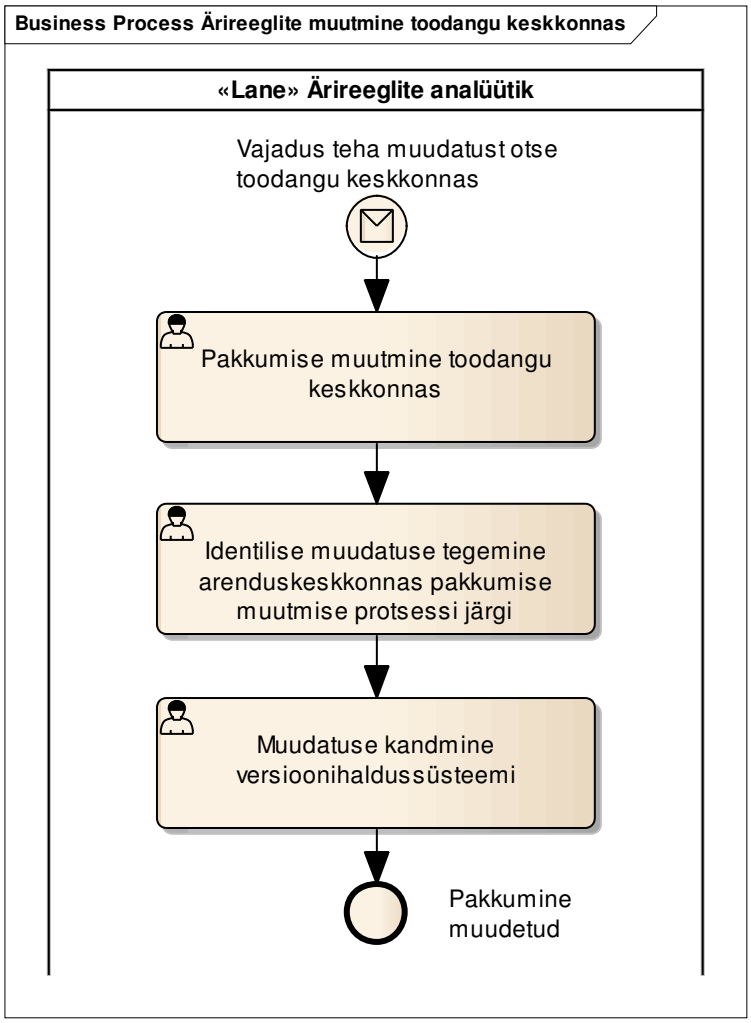


Joonis 6 Ärireeglite muutmine versioonivahetuse väliselt protsessi diagramm

Selleks, et mõnda reeglit muuta toodangu keskkonnas versioonivahetuse väliselt, on üheks võimaluseks teha arenduskeskkonnast eksport muudetud objektist ning importida see toodangu keskkonda. Pärast seda on vaja vajadusel värskendada ka reeglibaas, et reeglimootor oskaks uute reeglitega arvestada. Kui reeglibaasi värskendamine ebaõnnestub, siis jääb tööle vana reeglibaas.

Selline lähenemine ei õnnestu hästi, kui arenduskeskkonnas on tehtud mitu muudatust ning toodangu keskkonda soovitakse saada ainult osa nendest. Sellisel juhul tuleb käituda protsessi 3.8 „Ärireeglite muutmine toodangu keskkonnas“. (vt. Joonis 6)

3.8. Ärireeglite muutmine toodangu keskkonnas



Joonis 7 Ärireeglite muutmine toodangu keskkonnas

Ärireeglite muutmist otse toodangu keskkonnas võib vaja minna väikese muudatuse puhul, kui arenduskeskkonnas on sama objektiga tehtud suuremad muudatused ning neid ei soovita veel toodangusse kanda. Selline protsess peaks olema pigem erand, kuna võib põhjustada vigu toodangu keskkonnas. Autori praktilisest kogemusest on teada, et mõnikord läheb seda siiski vaja ja teatud muudatuste korral võib see olla põhjendatud. Antud protsessi järgi saab muuta ainult andmebaasis salvestatud pakkumiste häälestusi. Reeglifailide puhul peab siiski tegema muudatuse

arenduskeskkonnas. Soovi korral on võimalik võtta versioonihaldussüsteemist reeglifaili versioon, mis on aktiivne toodangu keskkonnas, muuta seda ning käituda protsessi 3.7 „Ärireeglite muutmine versioonivahetuse väliselt“. Kindlasti on oluline tagada samasugune pakkumise või reeglifaili muudatus ka arenduskeskkonnas ning kanda see versioonihaldussüsteemi. Vastasel juhul on oht, et järgmise versioonivahetusega kirjutatakse otse toodangukeskkonnas tehtud muudatused üle. (vt. Joonis 7)

4. Drools reeglimootori kasutamise prototüüp

Enne uue vahendi kasutusele võtmist suures ettevõttes, on kasulik proovida see läbi mõne väiksema näite peal. Kuidas antud vahend sobib püstitatud eesmärkidega ning millised probleemid või läbimõtlemata küsimused ilmnevad selle kasutamisel. Selles osas proovib autor järele, kas Drools reeglimootori kasutamine sobib Ettevõtte tootepakkumiste ärireeglite kirjeldamiseks. Selle jaoks realiseerib ta idee tõestamiseks *(*ingl.k. *proof of concept)* lihtsustatud reeglimootori kasutamise prototüübi, mis:

- oskab ehitada reeglibaasi,
- pakub teenuseid, mis käivitavad reeglimootorit sobival hetkel ning tagastavad reeglitega töödeldud tulemuse,
- oskab reeglibaasi uuendada ilma terve rakenduse tööd katkestamata.

Tehtud prototüübi abil saab kontrollida, kas kirjeldatud ärireeglid töötavad ootuspäraselt. Samuti aitab see kontrollida teisi mittefunktsionaalseid nõudeid.

Prototüübi abil soovib autor tõestada, et:

- Drools reeglimootori abil on võimalik mugavalt kirjeldada Ettevõtte tootepakkumiste ärireegleid,
- kirjeldatud reegleid on võimalik käivitada,
- käivitamise tulemus on korrektne ning kiire,
- ärireegleid on võimalik ilma versioonivahetuseta värskendada.

4.1. Prototüübi üldine kirjeldus

Tehtud prototüüp kujutab endast lihtsustatud veebirakendust, mille funktsionaalsus seisneb selles, et:

- veebiserveri käivitamisel ehitada reeglibaas,
- kuvada kasutajale vastavalt ärireeglitele tellimiseks lubatud tootepakkumised,
- lisada valitud tootepakkumisi ostukorvi,
- eemaldada valitud tootepakkumised ostukorvist,

- kuvada kasutajale ostukorvi lisatud tootepakkumised ja nende parameetrid ning muud ostukorviga seotud info,
- alustada ostukorvi kinnitamist.

Prototüüp on realiseeritud Java keeles ning kasutab järgmiseid teeke.

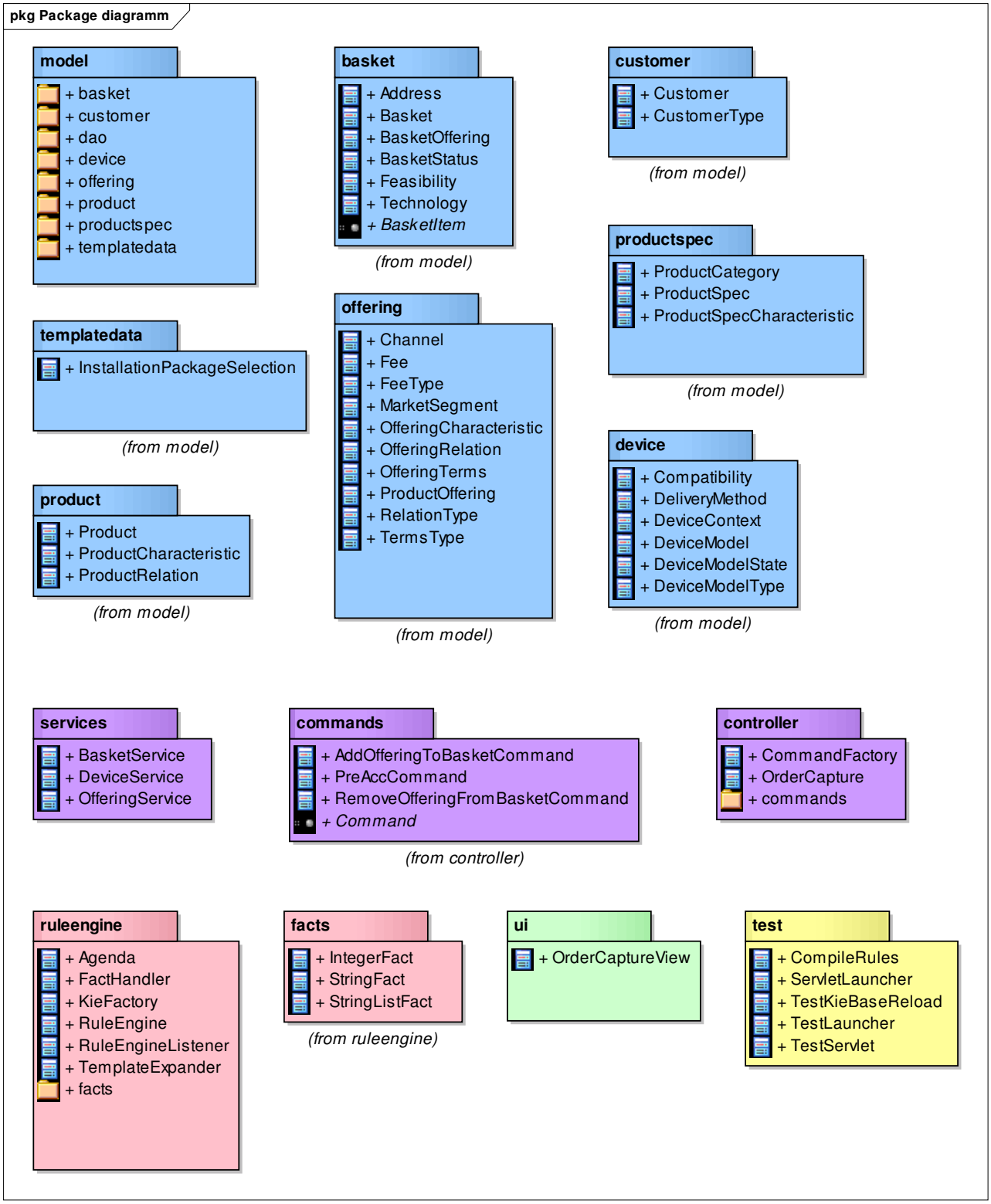
- JBoss Drools Expert 6.0
- Jetty 8.1.6
- Concordion 1.4.4
- JUnit 4.11
- Mockito 1.9.5
- SLF4J 1.7.2

Oluline on mainida seda, et lihtsuse huvides ei kasuta tehtud prototüüp andmebaasi. Kogu testimise jaoks vajalik info genereeritakse Java koodis ning sellega võltsitakse (ingl. k *mock*) kõik andmebaasi poole pöördumised.

4.2. Prototüübi paketidiagramm

Järgnevalt on toodud reeglimootori kasutamise prototüübi paketidiagramm (vt. Joonis 8). Diagrammil on esitatud olulisemad paketid ning klassid. Erinevaid paketi värve on kasutatud selle jaoks, et grupeerida pakette nende vastutusala järgi.

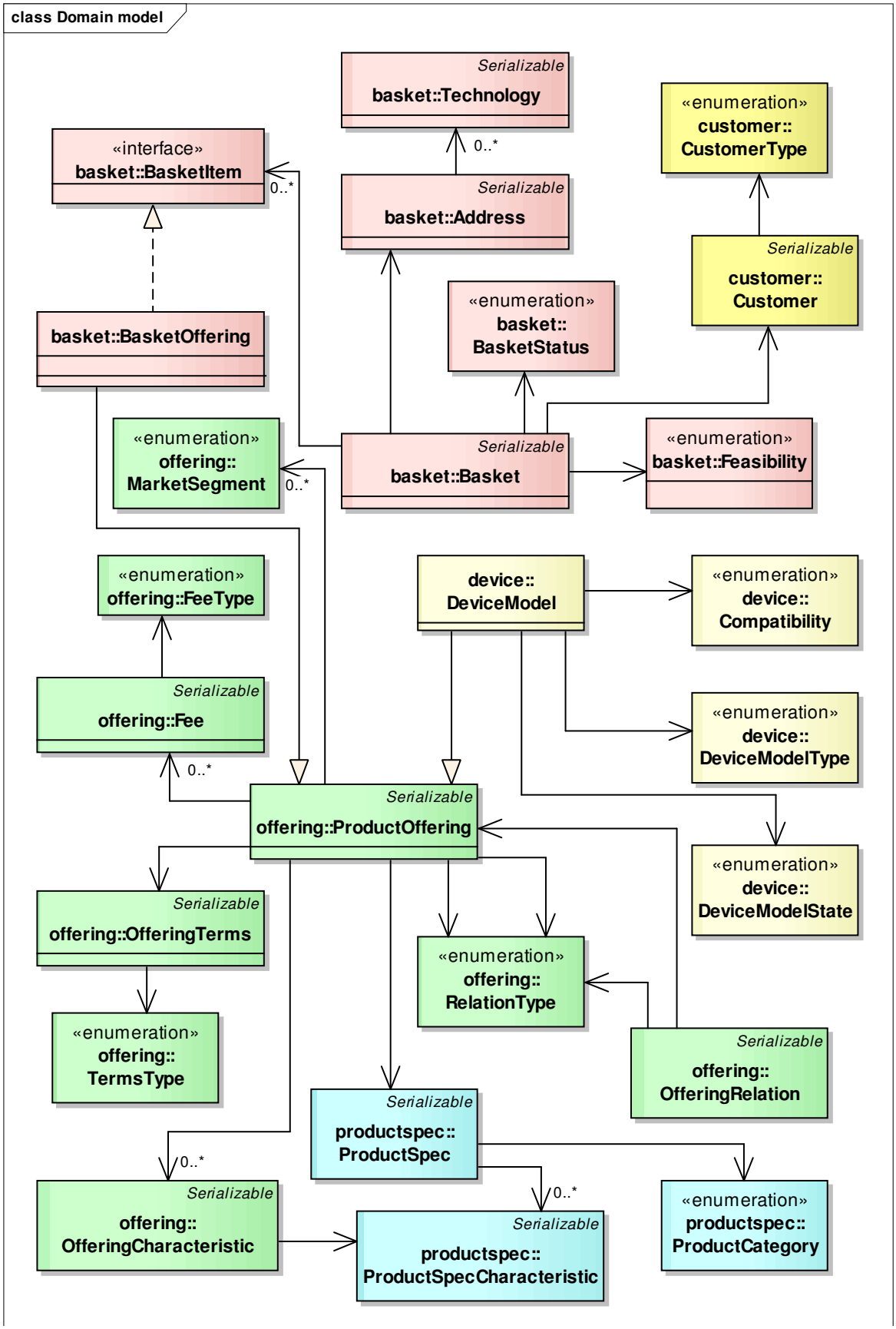
- Sinine – domeenimudeli klassid.
- Roosa – reeglimootori funktsionaalsus.
- Lilla – kontroller ning teenused.
- Roheline – kasutajaliidese joonistamine.
- Kollane – testimise jaoks vajalik funktsionaalsus.



Joonis 8 Prototüübi paketidiagramm

4.3. Prototüübi andmemudel

Järgnevalt on toodud prototüübis kasutatav andmemudel (vt. Joonis 9). Tegemist on lihtsustatud mudeliga, mis ei vasta üks ühele Ettevõttes kasutuses olevale andmemudelile, kuid mis on piisav selleks, et oleks võimalik kontrollida Drools reeglimootori toimivust ning ärireeglite kirjeldamise võimalusi. Andmemudeli aluseks on võetud TMForum grupi poolt välja töötatud kontseptuaalne andmemudel (*Information Framework (SID))*, mis on mõeldud spetsiaalselt telekommunikatsiooni ettevõtetele [19] ning mida järgitakse ka Ettevõttes.



Joonis 9 Prototüübi andmemudeli klassidiagramm

4.4. Reeglibaasi loomine

Drools pakub palju erinevaid võimalusi reeglibaasi loomiseks [11]. Selle töö raames valis autor niinimetatud programmilise variandi, kuna see on kõige paindlikum ning ei vaja Maven tööriista kasutamist.

Uue reeglibaasi *(KieBase)* loomine on ressursinõudlik protsess. Drools dokumentatsioonis soovitatakse võimaluse korral reeglibaasi kasutada korduvalt erinevate sessioonide loomiseks, kuna sessiooni tegemine on oluliselt kergem tegevus [11]. Antud prototüübis luuakse reeglibaas veebiserveri käivitamisel ning iga uue pöördumise korral luuakse reeglibaasi alusel uus sessioon.

Reeglibaasi loomine tähendab etteantud ressursside (reeglifailide, otsustustabelite) kompileerimist ning lisamist rakenduse mällu. Heaks tavaks on kontrollida pärast reeglibaasi loomist tekkinud vigu. Tehtud prototüübis vastutab reeglibaasi loomise ning tagastamise eest klass *ruleengine.KieBaseFactory.*

Oluline on märkida ka seda, et ühel rakendusel võib olla rohkem kui üks reeglibaas. Seda võib vaja minna näiteks juhul, kui soovitakse reegleid jaotada eraldiseisvamateks ning sõltumatuteks osadeks, mida oleks võimalik ükshaaval luua või muuta. Tuleb silmas pidada, et mitme reeglibaasi kasutamine sobib ainult siis, kui reeglibaaside reeglid on täiesti sõltumatud ja ei mõjuta üksteiste tööd, sest mitme baasi alusel ei ole võimalik luua ühist sessiooni reeglite käivitamiseks. Antud töös loob autor ainult ühe reeglibaasi, kuna tootepakkumise reeglid on omavahel sõltuvad ning taaskasutatavad mitme erineva pakkumise puhul.

4.5. Reeglite käivitamine

Reeglite käivitamiseks tuleb luua reeglibaasi alusel uus sessioon. Sessiooni tuleb lisada fakte (info, mille põhjal võib teha otsuseid). Prototüüpi tehes ja reegleid kirjeldades jõudis autor järelduseni, et mida lamedam on faktide struktuur, seda lihtsam on hiljem kasutada neid fakte reeglite tingimustes. Selle jaoks on autor teinud abiklassi, millega saab erinevaid objekte viia reeglimootori jaoks sobivamale kujule *(ruleengine.FactHandler).* Reeglite käivitamisel võib soovi korral määrata ka agendat, millega saab piirata käivitatavaid reegleid. Täpsemalt on antud töös kasutatavad agendad kirjeldatud peatükis 5.1.1.. Sama sessiooni käigus võib reegleid käivitada mitmekordselt. Lõpus tuleb aga sessioon kustutada. Sessioonile on võimalik lisada ka kuulajaid, mis logivad sessioonis

toiminud sündmusi (faktide lisamist, muutmist või eemaldamist, reeglite aktiveerimist ja käivitamist). Selline funktsionaalsus on väga kasulik silumise jaoks.

Üheks püstitatud nõudeks oli see, et ühe reegli käivitamise viga ei peataks tervet protsessi (NFR9). Täpsemalt öeldes, kui ühe reegli käivitamisel juhtub viga, peab reeglimootor siiski oma tööd jätkama ning käivitama kõik ülejäänud reeglid lõpuni. Reeglimootori väljakutsujal peab olema võimalus otsustada, kas konkreetses olukorras on antud viga oluline või mitte. Sellist käitumist võib saavutada kutsudes reeglite käivitamise välja tsükliliselt.

```
while( true ){
try {
       kieSession.fireAllRules();
       break;
      } catch(ConcequenceException e ){
            System.err.println( "### Runtime exception ###" );
            e.printStackTrace( System.err );
            }
        }
```

Muidugi ei tohiks viimati öeldu tähendada seda, et reeglimootori tingimustes juhtunud vigu võib ignoreerida. Vastupidi, neid peab analüüsima ning parandama. Antud nõue on jäänud töö sisse pigem põhjusel, et Ettevõttes kasutuses olev vana reeglimootor võimaldab sellist lähenemist.

Tehtud prototüübis vastutab reeglimootori sessiooni loomise, faktide lisamise, agenda määramise, reeglite käivitamise ning sessiooni kustutamise eest klass *ruleengine.RuleEngine*. Reeglimootori klassi kutsuvad välja erinevad teenuste klassid *(BasketService, OfferingService, DeviceService)*. Sarnast lähenemist kirjeldab Hakizumwami artiklis „Building Enterprise Services with Drools Rule Engine“[18].

4.6. Reeglibaasi uuendamine ilma katkestuseta rakenduse töös

Peatükis 4.3 rääkis autor, et kuna reeglibaasi loomine on ressursinõudlik protsess, on seda mõtet teha näiteks ühekordselt rakenduse käivitamisel ning hoida tulemust rakenduse mälus. See tähendab seda, et reeglifaili muutmine ei rakendu süsteemis koheselt, vaid ainult pärast seda, kui mälus hoitud reeglibaas laetakse ümber. Kusjuures uue reeglibaasi loomise protsessi käigus töötab vana reeglibaas jätkuvalt edasi. Alles siis, kui uue reeglibaasi loomine õnnestub, asendatakse süsteemis reeglibaasi (*KieBase)* objekti viide ning uued pöördumised kasutavad juba uut reeglibaasi.

Selle tõestamiseks tegi autor läbi järgmise katse (*test.java.TestKieBaseReload*).

1. Tehtud on kaks lihtsat reeglifaili: *before.drl* ning *after.drl.* Mõlemas failis olev reegel muudab sama ostukorvi parameetrit.
2. Testi käivitamisel luuakse reeglibaas esimese reeglifaili alusel.
3. Samal ajal luuakse üks eraldiseisev lõim *(*ingl.k *Thread),* mis iga kümne millisekundi tagant kontrollib sama ostukorvi parameetri väärtust.
4. Testi käigus laetakse reeglibaas ümber teise reeglifaili alusel.
5. Tulemuseks on näha, et lõimu töö ei olnud selle protsessi jooksul häiritud. Kui enne reeglibaasi uuendamist lõim tagastas parameetri ühte väärtust, siis pärast värskenduse õnnestumist väärtus muutus.

Tehtud katse tõestab seda, et on tõepoolest võimalik muuta reeglibaasi ilma katkestuseta süsteemi töös.

Kuna reeglibaas võib võtta palju rakenduse mälu, siis reaalse rakenduse tegemisel soovitab autor kindlasti olla tähelepanelik ning testida, et reeglibaasi uuendamisel ei tekiks rakenduses mälulekkeid selletõttu, et mõned objektid jäävad endiselt viitama vanale reeglibaasile.

5. Reeglibaas

Käesoleva töö ärireeglite haldussüsteemi prototüübi raames pidi autor looma ka esialgse reeglibaasi, ehk reeglid, mida oleks võimalik Drools reeglimootori abil käivitada. Loodud reeglibaas põhineb Ettevõttel kasutuses olevatel reeglitel. Kõige suurem rõhk on pandud kliendile pakutavate seadmete reeglite kirjeldamisele, kuna viimase aasta jooksul realiseeris autor samu reegleid oma põhitöö raames ka täna toimivas Ettevõtte süsteemis. Autori eesmärgiks oli kirjeldada võimalikult palju erinevaid reegleid kasutades seejuures mitmekülgseid lahendusi ning veendumaks selles, et Droolsi poolt pakutavate vahendite abil on võimalik kirjeldada kõiki Ettevõtte jaoks vajalikke reegleid.

5.1. Reeglite kirjeldamise põhimõtted

Autor tutvus ning õppis reeglite kirjeldamise süntaksit ning Drools vahendi eripärasid ja võimalusi. Selles peatükis kirjeldatakse põhimõtteid ning vahendeid, mida autor kasutas reeglite loomisel. Reeglite kirjeldamisel kasutas autor põhiliselt Drools veebilehel olevat dokumentatsiooni [10], kus on olemas nii sisukas seletus, kui ka head näited ning raamatut „JBoss Drools Business Rules“ [6].

5.1.1. Reeglite käivitamise koht

Reegleid võib vaja olla rakendada mitmes erinevas programmi kohas või protsessi sammus. Süsteemi üleliigse töö vältimiseks ei ole mõtet alati kontrollida kas kõikide reeglite tingimused on täidetud, vaid peaks välja valima ainult need reeglid, mis on konkreetses sammus olulised. Reegli rakendamise koha määramiseks kasutab autor reegli *agenda-group* atribuuti [12]. Reeglite käivitamisel antakse reeglimootorile ette agenda, mille reegleid peab käivitama.

Järgnevalt toob autor välja esialgse nimekirja agenda gruppidest, mida võib tootepakkumiste puhul vaja minna. Gruppide määramisel arvestas autor tänase Ettevõtte reeglimootori käivitamise kohtadega (pakkumisel, moodustamisel, salvestamisel, aktsepteerimisel) ning lisas juurde sellised grupid, mida autori arvates võib lisaks vaja minna või mida läks vaja juba rakenduse prototüübi tegemisel. Need grupid, mis on kasutuses ka rakenduse prototüübis, on märgitud rasvase kirjaga.

- ***init*** – reeglid, mida käivitatakse uue ostukorvi loomisel.
- ***offer*** – reeglid, mida käivitatakse pakkumiste pärimisel, selleks et filtreerida välja milliseid pakkumisi on lubatud pakkuda ja milliseid mitte.

- ***pre_add*** – reeglid, mida käivitatakse enne pakkumise ostukorvi lisamist.
- ***add*** – reeglid, mida käivitatakse pakkumise ostukorvi lisamisel.
- *save* – reeglid, mida käivitatakse ostukorvis pakkumise salvestamisel.
- *remove* – reeglid, mida käivitatakse ostukorvist pakkumise eemaldamisel.
- ***pre_acc*** – reeglid, mida käivitatakse enne kinnitamise protsessi alustamist.
- *accept* – reeglid, mida käivitatakse ostukorvi kinnitamisel.
- ***device*** – seadmetega seotud reeglid, mida tuleb käivitada vajadusel. Agenda määratakse teiste reeglite tegevustes.
- *change* – reeglid, mida käivitatakse kliendi toote muutmisel.
- *terminate* – reeglid, mida käivitatakse kliendi toote lõpetamisel.

5.1.2. Atribuuti *activation-group* kasutamine

Atribuuti *activation-group* võib vaja minna juhul, kui teatud reeglid on üksteist välistavad, mis teisisõnu tähendab, et sama grupi sees ühe reegli käivitamisel ei käivitata enam ülejäänud gruppi kuuluvaid reegleid isegi siis, kui nende tingimused on tõesed. Selline välistav reeglite käivitamine teeb väga oluliseks reeglite käivitamise järjekorra. Seda on võimalik juhtida atribuuti *salience* abil, millega on võimalik määrata reeglite prioriteeti. Sellises kombinatsioonis rakendub samast grupist ainult kõige prioriteetsem (*salience* on kõige suurem) reegel.

Autori jaoks osutus probleemiks see, kui gruppi kuuluva reegli tingimuse muster sobib mitme pakkumise jaoks, siis *activation-group* atribuudi kasutamisel rakendub see reegel ainult ühekordselt. Oodatud tulemus antud olukorras oleks aga see, kui üks gruppi kuuluvatest reeglitest rakendub iga pakkumise jaoks, mis reeglite tingimuse mustrit rahuldab. Seetõttu ei kasutanud autor atribuuti *activation-group* kirjeldatud reeglites.

5.1.3. Reegli tingimused

Reeglite tingimuste kirjeldamisest annab hea ülevaate Droolsi dokumentatsioon [13]. Erinevate tingimuste valik on väga suur ning reeglite kirjeldaja jaoks tähendab see seda, et suurem vabadus nõuab ka suuremat vastustust, sest tõenäosus teha midagi valesti on suurem. Autori saadud kogemusest reeglite kirjutamisel võib tuua välja järgmised tähelepanekud ning hoiatused.

- Tingimuste kirjutamisel peab olema ettevaatlik, et mitte tekitada lõputuid tsükleid. Need võivad tekkida näiteks juhul, kui tingimuses kontrollitakse objekti parameetrit, mida muudetakse sama reegli tegevuses. Lõputute tsüklite vältimiseks aitab atribuudi *no-loop* kasutamine või Java annotatsioonide *@PropertyReactive* ning *@Modifies* kasutamine, millest räägitakse täpsemalt järgmises alamjaotuses.
- Selleks, et reeglimootori töö oleks optimaalne, on kasulik pärast uute reeglite kirjutamist üle vaadata reeglite käivitamise logi, mida võib saavutada sessioonile kuulajate lisamisega. Logist on näha, milliste reeglite tingimused ja mis järjekorras on kontrollitud ning millised reeglid on lõpuks ka käivitatud (vt. täpsemalt peatükist 4.5).
- Tingimuste kirjutamisel tuleb veenduda, et otsitav muster ei oleks liiga üldine. Vastasel juhul võivad tingimustes olevate mustrite vahel tekkida suured ristkorrutised ning reeglimootor teeb palju ebavajalikku tööd. Hea näide selle kohta on toodud Droolsi dokumentatsiooni peatükis „6.1.4. Cross Products“[10].
- Autor üritas hoiduda ressursinõudlike Java meetodite väljakutsest (tingimuslik element *eval*) otse reegli tingimustes. Selle asemel lisas autor vajadusel puuduolevat infot uute faktidena reeglimootori sessiooni ning reegli tingimustes kasutas juba sessioonis olevaid fakte. Selline lähenemine annab suurema võidu siis, kui tegevusega lisatud fakti kasutatakse mitme erineva reegli tingimuses. Drools dokumentatsiooni peatüki 7.8.3.8 „Conditional Element eval“ järgi, saab reeglimootor selliselt oma tööd targemalt optimeerida [13].

5.1.4. Reegli tegevused

Reeglite tegevuste kirjeldamisest annab ülevaate Drools dokumentatsiooni peatükk 7.8.4., mis on pühendatud just erinevatele tegevustele, mida võib reeglites kasutada [14]. Tuleb mainida seda, kui tingimuste kirjutamisel saab kasutada väga palju Droolsi spetsiifilist süntaksit, siis tegevustes võib sisuliselt kirjutada tavalist Java koodi. Autor kasutab antud töös reegli tegevuste (tagajärgede) blokki peamiselt järgmistel eesmärkidel.

Fakti muutmine

Fakti muutmine on üks levinumatest toimingutes, mida autor reegli tegevustes kasutab. Selle toimingu abil võib näiteks muuta pakkumise parameetrit, mis juhib seda, kas pakkumist tohib pakkuda või mitte. Teine võimalus on muuta mingi pakkumise parameetri lubatud väärtusi vms. Seejuures on oluline aru saada, et fakti muutmist võib teha kahte erinevat moodi.

- Muuta Java objekt selliselt, et kehtiv sessioon ei saa sellest teada ning fakti muutmine ei põhjusta uute reeglite aktiveerimist ja käivitamist.
- Kui muudetud objekt peab mõjutama teisi reegleid, siis tuleb selle muutmisest anda teada ka reeglimootori sessioonile kasutades *modify* meetodit. Selleks, et anda reeglimootorile märku sellest, et muutunud ei ole terve objekt, vaid ainult selle teatud osa tulevad abiks annotatsioonid @*modifies* ja @*PropertyReactive* [13].

Uute faktide lisamine (konteksti loomine teiste reeglite jaoks)

Sellist võtet kasutab autor, kui teatud reeglid vajavad oma tingimustes täiendavat konteksti (fakte), mida ei ole reeglite käivitamisel sessiooni lisatud. Sellisel juhul arvutavad vajalikku lisakonteksti oma tegevustes teised reeglid ning lisavad sessiooni uusi fakte, millega ülejäänud reeglid saavad arvestada. Autori arvates on selline lähenemine parem, kui kõik vajalik konteksti lisamine juba enne reeglite käivitamist. Seda seetõttu, et selliselt lähenedes on võimalik lisada täiendavat konteksti vajaduse põhiselt mingite reeglite alusel. Näiteks ei pea süsteem arvutama seadmete reeglite jaoks vajalikku konteksti, kui seadmeid ei ole üldse vaja.

Agenda valimine

Reegli tegevuses on võimalik määrata fookus konkreetsele agendale *drools.setFocus* meetodi abil. See tähendab, et järgmisena kontrollib reeglimootor just etteantud agenda reegleid ning seejärel suundub tagasi parasjagu pooleli jäänud agenda juurde. Sellist meetodit on kasulik rakendada juhul, kui teatud reegleid on vaja kontrollida ainult mingite konkreetsete tingimuste täitumisel või kui samu reegleid on vaja kontrollida erinevates protsessi sammudes, millel on erinev agenda. Näiteks kasutab autor sellist lahendust reeglifailis *deviceModelRules.drl*, kus seadmete sobivuse reegleid tuleb arvutada ainult juhul, kui eksisteerib seadme pakkumine.

Ostukorvist objektide lisamine/eemaldamine

Osades reeglites pöördub autor teenuste poole, millega lisatakse või eemaldatakse pakkumisi ostukorvist. Näiteks võib sellist tegevust vaja minna selliste ärireeglite puhul, kus äriliselt soovitakse teatud reeglite alusel midagi automaatselt ostukorvi lisada või sealt eemaldada. Autor kasutab sellist lähenemist reeglifailis *themePackRules.drl*.

Teadete/vigade lisamine

Mõnikord on vaja teatud tingimustel kuvada kasutajale hoiatustekste või vigu. Kui tegemist ei ole üldreegliga, mis kehtib alati ja kõikide pakkumiste jaoks, siis on mõistlik selliseid hoiatusi ja vigu tõsta reegli tegevustesse, selleks et vältida erandite kirjutamist Java koodi. Näiteks kasutab autor sellist võtet reeglifailis *themePackRules.drl*.

5.1.5. Mallide kasutamine

Lisaks tavaliste reeglifailidele annab Drools võimaluse luua reeglite malle. See tähendab, et arendaja teeb valmis malli – *drt* formaadis faili, kuhu on jäetud järjehoidjad (ingl.k *placeholder*) mingisuguste andmete jaoks. Andmed ise võivad olla kirjeldatud näiteks andmebaasis. Reeglibaasi loomisel genereeritakse malli ning andmete alusel *drl* fail juba tavalisel kujul reeglitega. [15]

Selline lähenemine võib olla kasulik selliste ärireeglite puhul, kus teatud parameetrid muutuvad tihti ning äripoole inimesed soovivad neid ise hallata või kui eksisteerib hulk väga sarnase mustriga reegleid, kus ainsaks erinevuseks on sisendandmed. Autor proovis järgi reeglite mallide kasutamist reeglitel, mis asuvad failis *installationPackageSelectionRules.drt.*

Lisaks mallidele pakub Drools võimalust ka otsustustabelite (ingl.k *decision table*) kasutamiseks. Otsustustabelid sarnanevad oma omaduste poolest reeglite mallidega, kuid erinevuseks on see, et otsustustabelite sisu hoitakse *xls* või *csv* formaadis failides. Mallide kasuks otsustas autor just seetõttu, et oleks lihtsam hoida malli jaoks vajalikke andmeid andmebaasi tabelites, millele oleks kergem ehitada ka lihtsa kasutajaliidese nende andmete muutmiseks.

5.1.6. Kuidas tagada taaskasutatavus

Üheks käesoleva töö püstitatud nõudeks oli taaskasutatavuse nõue (NFR5). Näiteks kui on üle kahekümne erineva seadmepakkumise ning teatud ärireegel kehtib enamusele nendest, siis sellist ärireeglit peaks olema võimalik kirjeldada ühekordselt ning see peaks kehtima mitme seadmepakkumise jaoks. Selline reeglite taaskasutamine aitab vältida vastuolusid ärireeglites, vähendada töömahtu ning ka vigade allikat uute pakkumiste kirjeldamisel. Miinuseks taaskasutatavuse puhul võib välja tuua selle, et kui mitmes kohas kasutatavat reeglit muuta, siis see võib mõjutada paljusid pakkumisi. Üldjuhul on see vajalik, kuid mõnikord võib olla ka mitte ning tuleb arvestada sellise muudatuse ulatusliku mõjuga.

Järgnevalt toob autor välja meetodid, kuidas saab ärireegleid taaskasutada.

Üldistus läbi mustrite kasutamise

Reegli tingimuse muster võib olla nii üldine, et see rahuldaks kas või igat objekti reeglimootori sessiooni mälus [13]. Niimoodi saab kirjeldada piisava detailsusega mustrit, et see rahuldaks korraga mitut tootepakkumist, mille puhul peaks reegel kehtima. Näiteks võib selliselt üldistada erinevate seadmemudelite pakkumisi ning kirjeldada nende jaoks ühine reegel. Uue seadmepakkumise lisamisel rakenduvad talle kohe ka kõik seadmepakkumiste jaoks kirjeldatud reeglid. Sellist lähenemist kasutas autor näiteks seadme tarneviisi reeglis *channelTechnician*, mis asub reeglifailis *deliveryMethodRules.drl*.

Reeglite laiendamine

Drools pakub võimalust reeglite laiendamiseks. See võimalus ei ole väga hästi dokumenteeritud, kuid dokumentatsioonist võib siiski leida näiteid selle kasutamisest [16]. Kui üks reegel laiendab (ingl.k. *extends*) teist, siis see tähendab, et reegel rakendub ainult siis, kui nii selle, kui ka laiendatava reegli tingimused on täidetud. Reegli käivitamisel rakendatakse mõlema reegli tegevused. Selliselt võib tagada seda, et korduvad tingimused või tegevused on kirjeldatud ainult ühes reeglis. Näitena võib tuua reeglifailis *tvPlaceRules.drl* kirjeldatud reegleid *tvPlaces*, *tvPlaces4*, *tvPlaces2*, *tvPlaces1*. Sellise meetodi piiranguks võib välja tuua selle, et reegel, mis laiendab ning laiendatav reegel peavad asuma samas failis.

Tingimuslik reegli tegevus

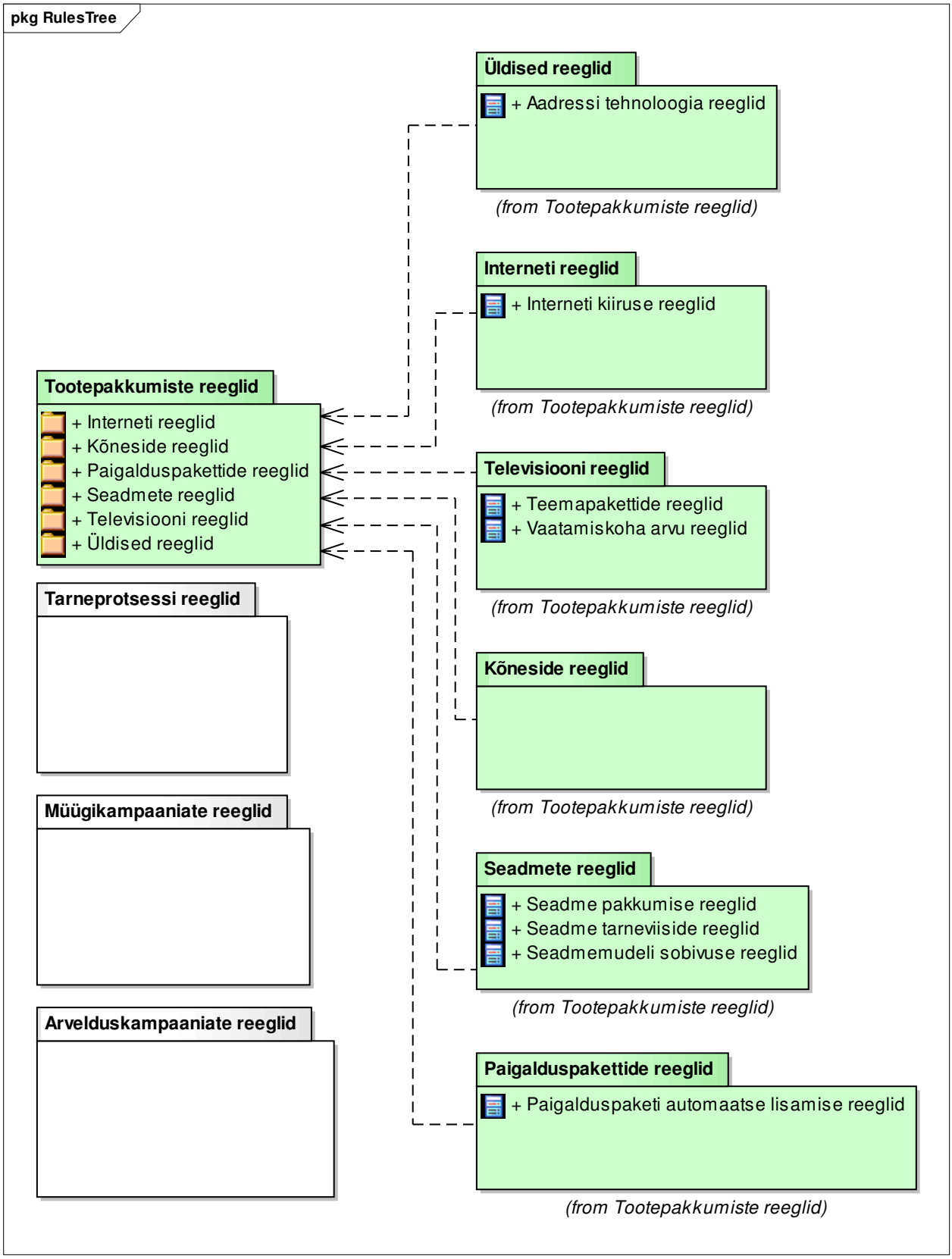
Lisaks pakub Drools võimalust ka selliste reeglite kirjeldamiseks, millel on mitu tegevuste plokki, kusjuures osadel plokkidel on täiendavad tingimused [16]. Sisuliselt saab selle funktsionaalsuse abil tagada sama tulemuse, mis ka reeglite laiendamisel. Tingimuslike tegevuste kasutamist proovib autor järele reeglifailis *deliveryMethodRules.drl* reeglites *installationPackageOrdered* ning *feasibilityLongOuting*.

5.2. Reeglibaasi struktuur

Selleks, et ärireeglid oleksid hallatavad ja lihtsalt leitavad, peavad nad olema struktureeritud ning loogiliselt jaotatud ja grupeeritud (NFR4). Kuna reeglid asuvad failisüsteemis erinevates failides,

siis nende grupeerimiseks kasutab autor erinevaid pakette (kataloogipuud). Kuna samad reeglid võivad mõjutada mitut erinevat pakkumist, siis grupeerimisel lähtus autor erinevatest teenuste valdkondadest (internet, kõneside, televisioon, seadmed) ning toote kategooriatest (interneti kiirus, teemapakett, vaatamiskoht).

Antud töös tehtud reeglite struktuur on esitatud joonisel 10. Rohelisega on esitatud realiseeritud reeglibaas ning valgena on lisatud juurde potentsiaalsed reeglibaasid, mida oleks võimalik tulevikus luua. Kindlasti ei ole tegemist lõpliku ega täieliku struktuuriga. Uute ärireeglite tekkimisel on ärireeglite arhitekti ülesandeks mõelda läbi reeglite asukoht olemasolevas struktuuris ning vajadusel reeglite struktuuri täiendada.



Joonis 10 Reeglibaasi struktuur

5.3. Kirjeldatud reeglid

Järgnevalt kirjeldab autor detailsemalt selle töö raames realiseeritud reegleid. Kuna ühe ärilise reegli realiseerimiseks võib vaja minna mitut Droolsi reeglit, siis parema ülevaate loomiseks koondab autor sama ärilise nõude eest vastutavad reeglid gruppidesse. Iga grupi juures on määratud:

- kategooria, kuhu reeglid reeglibaasi struktuuris kuuluvad,
- reegligrupi nimi,
- äriline kirjeldus,
- gruppi kuuluvate reeglite jaoks olulised sisendid,
- reeglite väljundid,
- asukoht, ehk reeglifaili nimi, kust võib leida gruppi kuuluvaid reegleid,
- agenda, mille raames reegleid käivitatakse,
- ning reeglite arv antud grupis.

Nagu ka eelmises peatükis oli mainitud, ei ole antud töös kirjeldatud kõiki Ettevõtte tootepakkumiste ärireegleid, vaid ainult osa nendest, mille alusel oleks võimalik hinnata Drools reeglimootori sobivust Ettevõtte reeglite kirjeldamiseks. Kokku on selle töö jooksul kirjeldatud 34 reeglit, mis on jaotatud 10 reegligrupi vahel. Võrdluseks võib öelda, et tänases süsteemis on reeglimootori reeglite arv ligikaudu 3000. Droolsi dokumentatsiooni peatükid 5.1.3 ning 5.4 väidavad, et süsteem skaleerub hästi suurte reeglite või faktide arvule [10].

Kategooria	Üldine
Reegligrupp	Aadressi tehnoloogia prioriteedi valik
Kirjeldus	Aadressil oletavatest tehnoloogiatest valib süsteem kõige prioriteetsema (pakutava) tehnoloogia, mille peale võib hakata midagi ehitama. Teised pakkumised arvestavad oma reeglites pakutava aadressi tehnoloogiaga.
Sisendid	• Kliendi poolt valitud aadress • Aadressil olevad tehnoloogiad • Tehnoloogiatevahelised välistavad seosed • Tehnoloogia prioriteet
Väljundid	• Pakutav tehnoloogia
Asukoht	*rules.offering.common.technologyRules.drl*
Agenda	*init*

Reeglite arv	3

Kategooria	Internet
Reegligrupp	Lubatud interneti kiirused
Kirjeldus	Sõltuvalt aadressil pakutavast tehnoloogiast filtreerib süsteem välja, milliseid kiiruseid tohib pakkuda.
Sisendid	• Aadressil pakutav tehnoloogia • Interneti kiiruste pakkumiste nimekiri • Tehnoloogiad, mille peal kiirus töötab
Väljundid	• Pakutavad interneti kiiruste pakkumised
Asukoht	*rules.offering.internet.internetSpeedRules.drl*
Agenda	*offer*
Reeglite arv	1

Kategooria	Internet
Reegligrupp	Interneti kiiruse vaikeväärtused
Kirjeldus	Interneti pakkumise tellimisel valib süsteem automaatselt vaikimisi interneti kiiruse pakkumise.
Sisendid	• Ostukorvis valitud interneti pakkumine • Ostukorvis valitud interneti kiiruse puudumine • Interneti kiiruse pakkumise prioriteet
Väljundid	• Interneti kiiruse pakkumine kõige suurema prioriteediga • Prioriteetseima interneti kiiruse pakkumise ostukorvi lisamine
Asukoht	*rules.offering.internet.internetSpeedRules.drl*
Agenda	*add*
Reeglite arv	1

Kategooria	Televisioon
Reegligrupp	Lubatud vaatamiskohtade arv

Kirjeldus	Sõltuvalt aadressil olevast tehnoloogiast ning sellest mitu vaatamiskohta on juba tellitud, otsustab süsteem kas on lubatud tellida veel vaatamiskohti juurde.
Sisendid	• Aadressil pakutav tehnoloogia
Väljundid	• Televisiooni pakkumise lubamine/mittelubamine • Lubatud vaatamiskohtade arv
Asukoht	*rules.offering.tv.tvPlaceRules.drl*
Agenda	*offer*
Reeglite arv	7

Kategooria	Televisioon
Reegligrupp	Teemapakettide vahelised reeglid
Kirjeldus	Teemapakettides Viasat Film, Viasat Sport ning Viasat Film ja Sport osa kanaleid kattuvad. Reeglitega juhitakse, et klient ei saaks valida enda jaoks rahaliselt kahjulikku kombinatsiooni nendest kolmest teemapaketist.
Sisendid	• Ostukorvis olevad teemapaketid • Ostukorvi lisatav teemapakett
Väljundid	• Veateade selle kohta, et teemapaketi lisamine ei ole lubatud • Ostukorvist teemapaketi eemaldamine • Ostukorvi teemapaketi lisamine
Asukoht	*rules.offering.tv.themePackRules.drl*
Agenda	*pre_add, add*
Reeglite arv	3

Kategooria	Seadmed
Reegligrupp	Seadmemudeli sobivuse reeglid
Kirjeldus	Reeglid, mis otsustavad, kas seadmemudel sobib või mitte etteantud kontekstiga (tellitud kiirus, tehnoloogia vms.).
Sisendid	• Valitud interneti kiirus • Seadmemudeli pakkumiste nimekiri • Televisiooni komponendi olemasolu • Aadressil pakutav tehnoloogia

Väljundid	• Sobiv mudel • Mittesoovituslik mudel • Mittesobiv mudel
Asukoht	*rules.offering.device.deviceModelRules.drl*
Agenda	*device*
Reeglite arv	7

Kategooria	Seadmed
Reegligrupp	Seadme pakkumise reeglid
Kirjeldus	Sõltuvalt sellest, kas pakkumise seadmemudel sobib või mitte ning kas pakkumise tingimused on lubatud või mitte, valib süsteem välja seadmepakkumisi, mida tohib pakkuda.
Sisendid	• Mudeli tüüp (ruuter /digiboks /telefon/ jne.) • Mudeli sobivus • Aadressil pakutav tehnoloogia • Pakkumise segment (äri/era) • Seadmepakkumise tingimused • Võimalikud seadmemudeli pakkumised
Väljundid	• Pakutavad seadmemudeli pakkumised
Asukoht	*rules.offering.device.deviceOfferingRules.drl*
Agenda	*offer*
Reeglite arv	4

Kategooria	Seadmed
Reegligrupp	Seadme vaikeväärtused
Kirjeldus	Seadme tellimisel valib süsteem automaatselt välja vaikimisi seadmemudeli pakkumise. Vaikimisi valik erineb sõltuvalt aadressi tehnoloogiast.
Sisendid	• Ostukorvis valitud seadmepakkumine • Ostukorvis valitud seadmemudeli puudumine • Aadressil pakutav tehnoloogia
Väljundid	• Seadmemudeli pakkumise lisamine ostukorvi

Asukoht	*rules.offering.device.deviceOfferingRules.drl*
Agenda	*add*
Reeglite arv	2

Kategooria	Seadmed
Reegligrupp	Seadme tarneviisi reeglid
Kirjeldus	Ostukorvi kinnitamise alustamisel arvutab süsteem välja seadmepakkumistele lubatud tarneviisi. Tarneviisi valik sõltub tehnilise selgituse vastusest, paigalduspaketi valikust ning tellimise kanalist.
Sisendid	• Sisendkanal • Seadmemudeli pakkumine • Automaatse tehnilise selgituse vastus • Ostukorvis paigalduspaketi pakkumise olemasolu
Väljundid	• Seadmemudeli pakkumise lubatud tarneviisid
Asukoht	*rules.offering.device.deviceDeliveryMethodRules.drl*
Agenda	*pre_acc*
Reeglite arv	5

Kategooria	Paigalduspaketid
Reegligrupp	Paigalduspaketi automaatne lisamine
Kirjeldus	Kui süsteem tuvastab, et kliendi seadmed ei pruugi uue lahendusega sobida, siis süsteem lisab automaatselt paigalduspaketi pakkumise. See tähendab, et tehnik läheb kliendi juurde igaks juhuks kohale.
Sisendid	• Kliendil võrgus olev seade • Kliendile lepinguga antud kehtiv seade
Väljundid	• Paigalduspaketi pakkumise lisamine ostukorvi • Teade teenindajale
Asukoht	*rules.offering.installationpackage.installationPackageRules.drl*
Agenda	*pre_acc*
Reeglite arv	1 (reeglimall)

5.4. Ärireeglite versioonihaldus

Ärireeglite versioonide haldamiseks pakub autor oma lahenduses kasutada versioonihaldustarkvara GIT. See on Ettevõttes levinud vahend ning seda kasutab ka praegune tootekataloogi rakendus. Lahendus toimib täna hästi ning on ennast juba ära tõestanud ja seetõttu pakub autor seda kasutada ka edaspidi. Lahenduse kohaselt hoitakse versioonihaldussüsteemis kõiki ärireegleid, mille hulka kuuluvad tootepakkumised, reeglifailid ning mallide jaoks vajalikud andmed. Selline lähenemine võimaldab:

- jälgida muudatuste ajalugu,
- tuvastada, kes tegi reeglis konkreetse muudatuse,
- taastada reeglifaili või pakkumise kindel versioon,
- paigaldada arenduskeskkonnas tehtud muudatused toodangu keskkonda, hoides seeläbi arenduse ja toodangu keskkondade reeglite seisu omavahel sünkroniseerituna.

Ärianalüütikud, kes üldjuhul muudavad pakkumisi andmebaasis, saavad kanda tehtud muutasi versioonihaldussüsteemi läbi abistava kasutajaliidese. Läbi kasutajaliidese on võimalik luua pakkumise eksportfail ning lisada ka muudatuse kommentaar. Genereeritud faili sisu koos kommentaariga salvestatakse andmebaasi tabelisse. Automaatne protsess tõmbab tabelisse salvestatud muudatused regulaarselt versioonihaldussüsteemi.

Eelnevalt kirjeldatud protsess eeldab, et pakkumiste eksportimiseks on loodud abistavad programmid, mille ülesanne on konverteerida pakkumise andmed, mis võivad asuda erinevates andmebaasi tabelites, ühtsesse kokkulepitud formaati (näiteks XML), mida saab failina lisada versioonihaldussüsteemi. Versioonipaki paigaldamisel peab saama moodustada selle faili alusel vajalikud SQL laused pakkumise sisu tagasi andmebaasi salvestamiseks. Kuna tootepakkumiste arv ühes süsteemis võib olla suur, siis peab pakkumiste granulaarsuseks versioonihaldussüsteemis olema üks pakkumine, mitte näiteks terve pakkumiste tabeli sisu. Selliselt on kõige mugavam jälgida pakkumise muudatuste ajalugu ning kanda pakkumise muudatused ühest keskkonnast teise.

Kui tootepakkumiste versioonimine käis analoogsel viisil ka vanas süsteemis, siis reeglimootoris käivitatavate reeglite versioonimine läheb käesoleva töö lahenduses lihtsamaks, sest reegleid ei hoita enam andmebaasis, vaid reeglifailides, mida on lihtsam versioonihaldussüsteemi lisada. Seda sellepärast, et puudub vajadus tabeli sisu konverteerimiseks failiks. Selliselt hoitud reeglid

säilitavad versioonihaldussüsteemis kirjeldaja jaoks tuttava kuju ja seetõttu on ka reeglifailis tehtud muudatuste jälgimine mugavam ning arusaadavam.

6. Ärireeglite testimine ning dokumenteerimine

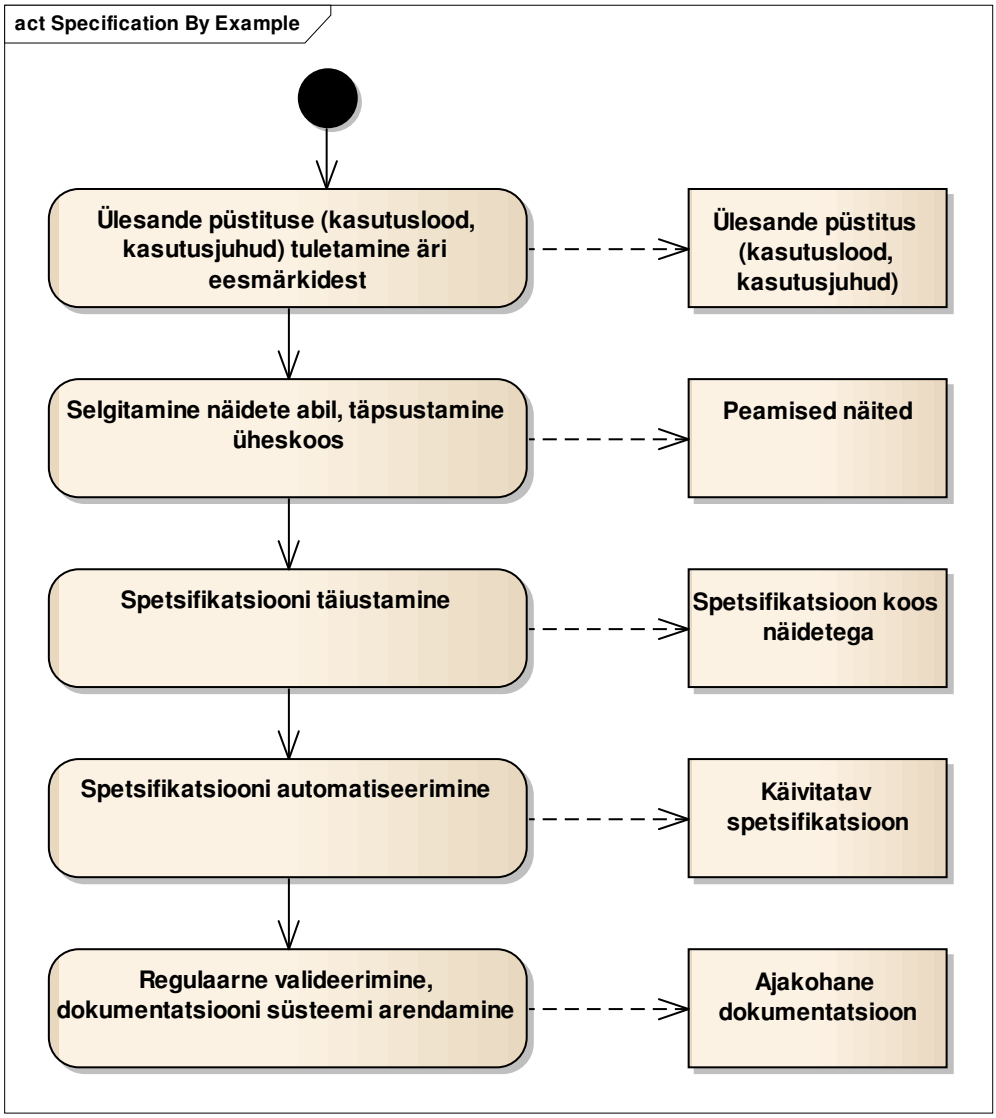
Üheks probleemiks, mida autor soovis antud tööga lahendada, on praeguses süsteemis realiseeritud ärireeglite läbipaistmatus ning dokumentatsiooni puudumine. Antud teemaga seoses püstitati ka nõue NFR2, mis rääkis sellest, et äritellijal peab olema koht, kus ta saab loetaval kujul vaadata süsteemis realiseeritud ärireegleid. Täna on ainus koht vaatamiseks tootepakkumiste reeglid ise, kuid nendest on tihti keeruline aru saada ja seepärast äripool neid ei vaata. Pealegi ei anna need ülevaadet sellest, kas äriliselt oligi niimoodi tellitud või on süsteemis lihtsalt midagi valesti kirjeldatud. Ainult reegleid vaadates ei saa olla kunagi kindel, kas süsteemis on realiseeritud äriliselt õiged reeglid või mitte.

Teiseks probleemiks oli see, et ühe reegli muutmine võib tihti mõjutada palju protsesse ning tekitada uusi vigu, mistõttu nõuab reeglite muutmine pidevat ja mahukat testimist. Selle mure lahendamiseks püstitati nõue, et kõik ärireeglid peavad olema kaetud automaattestidega (NFR8).

Lahenduseks, mis aitab leevendada mõlemat eelpool mainitud probleemi, pakub autor välja kasutada Adzici raamatus „Specification by Example“ kirjeldatud näidete abil kirjeldamise meetodit. Analoogset meetodit tuntakse ka teiste nimedel all, nagu näiteks väle vastuvõtutestimine (ingl.k. *Agile Acceptance Testing*), vastuvõtutestimiskeskne arendus (ingl.k *Acceptance Test-Driven Development*), loo teostuse testimine (ingl.k *Story Testing)* ja teised. [1, preface]. Siinses peatükis kirjeldatakse seda meetodit ning antud töö raames selle kasutamist detailsemalt.

6.1. Põhimõtted

Kokkuvõtlikult öeldes seisneb meetodi idee selles, et süsteemi dokumentatsioon kasvab välja spetsifikatsioonidest (antud töö puhul - kirja pandud ärireeglitest), mis on kaetud näidete alusel tehtud automaattestidega. Sellist kooslust nimetatakse käivitatavateks spetsifikatsioonideks. Antud meetod koosneb järgmistest sammudest ning tehistest (vt. Joonis 11) [1, lk 18].



Joonis 11 Näidete abil kirjeldamise meetodi sammud ja tehised

Adzic rõhutab, et eelmises punktis toodud sammud on iteratiivsed. See tähendab, et sellist protsessi ei tehta korraga kõikide spetsifikatsioonide jaoks, vaid spetsifikatsioonid võetakse töösse ainult siis, kui meeskond on valmis neid teostama hakkama (näiteks uue iteratsiooni alguses).

Järgnevalt tuuakse välja eelised, mis autori arvates kaasnevad näidete abil kirjeldamise meetodi kasutamisega.

- Äripool ning meeskonna liikmed on kaasatud spetsifikatsioonide loomisel ning näidete kirjeldamisel, mis aitab üheselt aru saada ärivajadusest ning selgitada välja kitsaskohti.

- Ärireeglite dokumentatsioon on alati värske ning kooskõlas süsteemi realisatsiooniga, kuna see on kaetud automaattestidega. Kui dokumentatsioon peaks minema kooskõlast välja, siis automaattestid annavad sellest koheselt ebaõnnestumisega märku ja neid tuleb parandada.
- Ärireeglite dokumentatsioon on kõikidele osapooltele kättesaadav ning on arusaadav.
- Ärireeglite muudatuste sisseviimine on lihtsam, kiirem ja turvalisem, kuna automaattestid loovad kindluse, et vana funktsionaalsus jääb tööle.

6.2. Vahendid

Tänapäeval on olemas mitu vahendit, mis aitavad spetsifikatsioone automatiseerida. Ühed populaarsemad nendest on näiteks Cucumber, FitnessWiki ja Concordion. Kindlasti ei ole vahendi valik selle meetodi puhul kõige olulisem aspekt, kuid siiski aitab hea vahend mugavamalt jõuda soovitud tulemuseni. Käesolevas töös valis autor käivitatavate spetsifikatsioonide loomiseks vahendi Concordion [9], kuna sellel on, autori arvates, kõige kasutajasõbralikum ja loetavam väljund ning see on ka väga paindlik.

Concordioni miinuseks võib välja tuua selle, et see nõuab spetsifikatsiooni kirjeldajalt HTML'i oskust, mis tähendab, et kirjeldajal peab olema vähemalt algtasemel teadmised HTML'ist. Samuti võib see kaasa tuua ka töömahu suurenemise testi kirjutamisel. Autori meelest, on need miinused ületatavad. Näiteks võib alguses kirjeldada spetsifikatsioone lihtsalt tekstilisel kujul ning hiljem testi kirjutaja instrumenteerib seda vajalike HTML siltide ja atribuutidega. Valmistehtud mallid käivitatavatest spetsifikatsioonidest aitavad aga vähendada üleliigset tööd ning tagavad kooskõla kirjeldatud spetsifikatsioonide vahel.

Vahendi kasutamise põhimõte seisneb selles, et iga ühiktesti klassile *(Fixture class)* vastab ka üks spetsifikatsioon, mis kujutab instrumenteeritud HTML faili, kus kasutatakse spetsiaalseid Concordion siltide atribuute:

- *concordion:set* – väärtuse määramiseks,
- *concordion:assertEquals* – väärtuse kontrollimiseks,
- *concordion:assertTrue* – tõeväärtuse kontrollimiseks,
- *concordion:execute* – meetodi käivitamiseks,
- *concordion:verifyRows* – massiivi väärtuste kontrollimiseks.

Ühiktestide käivitamisel genereerib süsteem automaatselt dokumentatsiooni ettemääratud kausta, milles on näha ka testide tulemused. Näiteid genereeritud dokumentatsioonist leiab järgmisest peatükist või lisast 2.

Lisaks Concordionile ning JUnit raamistikule kasutab autor ka Mockito testimise raamistikku. Sarnast kombinatsiooni reeglite testimiseks pakub ka Tirelli [26]. Ärireegleid on väga mugav testida nii, et andmebaasi kiht on asendatud nö võltsobjektidega *(*ingl.k *mock)*. See annab võimaluse valmistada käsitsi ette sobilikud sisendandmed ning testida lihtsalt läbi mitu erinevat kombinatsiooni.

6.3. Käivitatav spetsifikatsioon

Antud töö raames on spetsifitseeritud ning automatiseeritud 5 ärilist nõuet, mis katab umbes poole tehtud reeglitest. Näiteks võib tuua seadmete tarneviisi reeglite spetsifikatsiooni, mille puhul näeb automaattesti tulemusena genereeritud dokumentatsioon välja järgmiselt (vt. Joonis 12). Jooniselt on näha, et lehe ülemises osas on olemas ka jäljerida (ingl.k. *breadcrumb*), mida genereeritakse automaatselt ning mille abil on mugav dokumentatsiooni sees navigeerida. Lehe alumises paremas nurgas on näha, millal oli dokumentatsioon genereerid, ehk millal oli automaattest viimati käivitatud. Testide sisendid on märgitud rasvases kirjas ning testi tulemused on näha vastavalt rohelistes või punastes kastides, sõltuvalt sellest kas test õnnestus või mitte. Lisas 2 tuuakse rohkem näiteid tehtud käivitatavate spetsifikatsioonide kohta.

Ärireeglite dokumentatsioon > Pakkumiste ärireeglid > Seadmepakkumiste ärireeglid >

Tarneviisi valiku reeglid

1. Reeglid ruuteritele, digiboksidele ning IP telefonidele:
    - Tehnikuga tarneviis on lubatud, kui kehtib vähemalt üks järgnevatest lausetest:
        - on tellitud paigalduspakett,
        - tehnilise selgituse vastus on pikk tehniline selgitus või väljasõit,
        - tellimise kanal on Anto.
    - Esinduse kanalist on alati lisaks Tehnikuga tarneviisile lubatud ka tarneviis Esindusest.
    - Tarneviisid Esindusest ja Kulleriga on lubatud esinduse ja veebi kanalist siis, kui tehnikuga tarneviis ei ole lubatud.
2. Terminale tarnitakse alati ainult Tehnikuga.

Näited

1. Äri interneti tähtajalise kasutuse Inteno D301 ruuteri (**401**) pakkumise puhul kehtivad jägmised reeglid:

Kanal	Paigalduspakett tellitud	Tehnilise selgituse vastus	Lubatud tarneviisid	Tarneviiside arv
Veeb (**WEB**)	**Jah**	Auto (**AUTO**)	Tehnikuga (TECHNICIAN)	1
Esindus (**STORE**)	**Jah**	Auto (**AUTO**)	Tehnikuga (TECHNICIAN) , Esindusest (STORE)	2
Veeb (**WEB**)	**Ei**	Auto (**AUTO**)	Esindusest (STORE) , Kulleriga (COURIER)	2
Anto (**TECH**)	**Ei**	Auto (**AUTO**)	Tehnikuga (TECHNICIAN)	1
Veeb (**WEB**)	**Ei**	Pikk tehniline selgitus (**LONG**)	Tehnikuga (TECHNICIAN)	1
Esindus (**STORE**)	**Ei**	Väljasõiduga (**OUTING**)	Tehnikuga (TECHNICIAN) Esindusest (STORE)	2

2. Interneti tavakasutuse terminali D12356 (**428**) pakkumise puhul kehtivad jägmised reeglid:

Kanal	Paigalduspakett tellitud	Tehnilise selgituse vastus	Lubatud tarneviisid	Tarneviiside arv
Veeb (**WEB**)	**Ei**	Auto (**AUTO**)	Tehnikuga (TECHNICIAN)	1

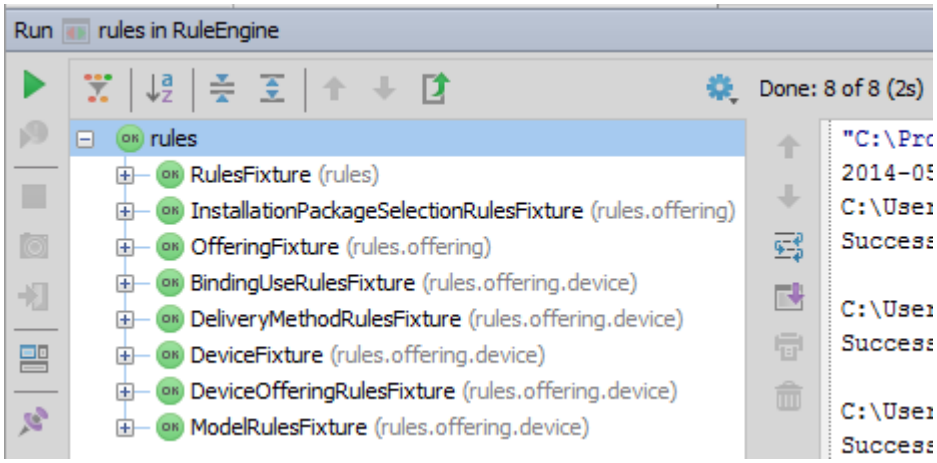
Edasi

- TODO: Mis saab, kui paigalduspakett lisatakse ja seejärel eemaldatakse?

Results generated by Concordion
in 654 ms on 18-apr-2014 at 17:53:49 EEST

Joonis 12 Käivitatava spetsifikatsiooni näide

Automaattestide tulemusi on võimalik näha mitte ainult Concordioni poolt genereeritud dokumentatsioonist, vaid ka tavalise automaattestide käivitamise tulemusena (vt. Joonis 13).



Joonis 13 Automaattestide käivitamise tulemused IDE's

7. Vastavus püstitatud nõuetele

Peatükis 2.1 püstitab autor peamisi mittefunktsionaalseid nõudeid ärireeglite haldussüsteemile. Tehtud prototüübi, reeglibaasi ning dokumentatsiooni alusel võib hinnata vastavust püstitatud nõuetele. Järgnevast tabelist (vt. Tabel 2) on näha, et tehtud rakenduse prototüüp rahuldab kõiki püstitatud nõudeid.

Nõue	**Tulemus**	**Kommentaar**
NFR1	OK	JMeter'i abil oli viidud läbi koormustest 20 paralleelse lõimega, kus umbes 5 sekundi jooksul töötles süsteem kokku 1000 pöördumist. Pöördumise keskmiseks ajaks oli 42 millisekundit ning ükski pöördumine ei saanud veateadet, mis on väga hea tulemus. Täpsemat testi tulemust saab vaadata lisast 3. Kindlasti ei ole antud testi käivituskeskkond võrdeline reaalse süsteemiga ning on erinev ka reeglite arv reeglibaasis, kuid tulemus näitab seda, et prototüübi puhul koormusega seotud probleeme ei tekkinud.
NFR2	OK	Dokumentatsioon käivitatavate spetsifikatsioonide kujul on see koht, kust saab loetaval kujul näha süsteemis realiseeritud ärireegleid.
NFR3	OK	Äripoole inimesed saavad ise teha lihtsamaid muudatusi ärireeglites, mida saab häälestada pakkumiste kirjelduste ning reeglimallide andmete abil ning mida hoitakse andmebaasis. Pakkumiste kirjeldamise protsess kirjeldatakse peatükis 3.4. Reeglimootori poolt käivitatavad reeglid ei kuulu vähemalt esimeses realisatsioonis lihtsamate ärireeglite hulka. Tulevikus on siiski võimalik kaaluda võimalusi, kuidas ka selliselt kirjeldatud reegleid oleks võimalik muuta ilma tehniliste teadmisteta inimestel. Mõned ideed selle kohta on toodud välja peatükis 2.3.1 ning töö kokkuvõttes.
NFR4	OK	Ärireeglite võimalik struktuur kirjeldatakse peatükis 5.3
NFR5	OK	Taaskasutatavuse võimalused tuuakse välja peatükis 5.1.6
NFR6	OK	Ärireeglid on versioonitavad GIT versioonihaldussüsteemi abil. Põhimõtted tuuakse välja peatükis 5.5
NFR7	OK	Selle nõude täidab Droolsi sisseehitatud funktsionaalsus erinevate kuulajate näol,

		mis on täpsemalt kirjeldatud peatükis 5.4.
NFR8	OK	Ärireeglite automaattestimiseks pakutakse lahendust peatükis 6.
NFR9	OK	Lahendus pakutakse välja peatükis 4.4.
NFR10	OK	Protsess kirjeldatakse peatükis 3.6.
NFR11	OK	Protsess kirjeldatakse peatükkides 3.7 ning 3.8.
NFR12	OK	Nõude tagavad protsessid, mis on kirjeldatud peatükkides 3.4 - 3.8.

Tabel 2 Vastavus nõuetele

Kokkuvõte

Käesoleva töö peamiseks eesmärgiks oli pakkuda välja lahendus Ettevõtte tootepakkumiste ärireeglite haldusega seotud protsesside ning süsteemide parendamiseks.

Töö käigus lõi autor kontseptsiooni, milline võib olla Ettevõtte jaoks sobiv pakkumiste ärireeglite haldussüsteem. See kontseptsioon hõlmab endas olulisemaid mittefunktsionaalseid nõudeid, parendatud ärireeglite haldamise protsessi kirjeldust, ärireeglite kirjeldamise, versioonimise, paigaldamise, testimise ning dokumenteerimise põhimõtteid. Kontseptsiooni tõestamiseks lõi autor reaalse süsteemi prototüübi, mille põhiliseks vahendiks on Drools reeglimootor ning kirjeldas ka esialgseid reegleid. Kokku on selle töö raames kirjeldatud 34 reeglit, mis sisuliselt lahendavad 10 erinevat ärilist probleemi, millest 5 olid ka spetsifitseeritud ning automatiseeritud kasutades näidete abil kirjeldamise meetodit. Tehtud prototüübi alusel võib väita, et püstitatud nõuded olid edukalt täidetud ning lahendus sobib reaalses süsteemis rakendamiseks.

Järgnevalt toob autor välja Drools reeglimootori kasutamisega seotud järeldused.

- Oluline on aru saada Drools reeglimootori töö põhimõtetest ning jälgida kirjeldatud reeglite käivitamise järjekorda. Vastasel juhul on lihtne mitte märgata ebaoptimaalset reeglite kirjeldust ning vigu, mis võivad põhjustada toodangu keskkonnas ootamatut süsteemi käitumist ning koormuse probleeme.
- Drools reeglimootor on väga paindlik ning annab reeglite kirjeldajale palju vabadust. Ühest küljest on see hea, kuid teisest küljest lisab see suurema vastutuse, kuna võimalus eksida on samuti suurem.
- Reeglimootori kasutamisega on automaattestidel väga oluline roll, et tagada süsteemi stabiilsus ning töökindlus.

Edasisteks sammudeks selle töö puhul on võtta Drools reeglimootor kasutusse reaalses süsteemis ning juurutada täiendatud ärireeglite haldamise protsessid. Tehtud prototüübis oli reeglimootor ning reeglibaas – üks osa tellimissüsteemi rakendusest. Üheks alternatiiviks on realiseerida tsentraalne ärireeglite haldussüsteem. Sellisel juhul asuksid kõik ärireeglid ühes tsentraalses reeglite hoidlas ning ülejäänud süsteemid, mis soovivad reeglimootorit kasutusele võtta, ei peaks sel juhul hakkama uuesti otsast peale välja töötama kogu reeglite kirjeldamise, paigaldamise, testimise ning

dokumenteerimise loogikat, vaid peaksid oskama vaid oma rakendusest reeglibaasi ehitada ning õiges kohas reeglimootorit välja kutsuda.

Lisaks sellele võib kaaluda valdkonna spetsiifilise keele (ingl.k *Domain Specific Language*) arendamist, mis oleks mõeldud spetsiaalselt Ettevõtte valdkonna reeglite kirjeldamiseks [2, lk. 82 - 92]. Selle juurde oleks võimalik arendada ka spetsiaalne kasutajaliides, mille kaudu saaksid lihtsamaid ja levinumaid reegleid kirjeldada ka ilma tehnilise hariduseta inimesed. Kindlasti vajab edasiarendust ka dokumentatsiooni lahendus. Arengusuunaks selles osas on leida dokumentatsioonile sobiv koht, kust see oleks kõikidele huvitatud osapooltele kättesaadav ning dokumentatsiooni mahu suurenemisel mõelda läbi otsingu võimalused.

Summary

The main purpose of this thesis was to propose a solution to problems related to product offerings business rules management processes and systems in the Enterprise.

During this work the author has created a concept, how a suitable product offerings business rule management system for the Enterprise should look like. This concept comprises the most important non-functional requirements, improved management process description and principles of writing, deploying, testing and documenting business rules. As a proof of concept the author has created a system prototype using Drools rule engine and described initial rules. 34 rules in total were described, which solve 10 different business problems. 5 of them were also specified and automated using "Specification By Example" method. The created prototype met all set requirements successfully and that gives reason to believe that the solution is suitable to be used in the actual system.

From the experience of using Drools rule engine the following conclusions can be made.

- It is important to understand the concepts of Drools rule engine and track the execution order of rules. Otherwise it is easy not to notice non-optimal rule descriptions and errors, which can cause unexpected behavior and performance issues in the production system.
- Drools rule engine is very flexible and gives a lot of freedom to the rules writer. On the one hand, it is a good property. On the other hand, it adds more responsibility, because it is possible to make more mistakes.
- The role of automated tests is extremely important while using a rule engine to ensure a stable and robust system.

The next steps regarding this work would be to implement the real system using Drools rule engine and to introduce improved business rule management processes in the Enterprise. A rule engine was a part of the order capture system in the prototype. One alternative approach is to create a central business rule management system. In this case, all business rules would be placed in the central rule repository and the rest of the systems, which would like to use a rule engine, would not need to develop the whole rules description, deployment, testing and documentation logic themselves. They would just need to know how to build a rule base and apply rule engine where needed.

In addition, a development of the Enterprise domain specific language should be considered [2, p. 82 - 92]. This would give an opportunity to develop a custom user interface for describing the most common rules for people without technical skills. Documentation solution can also be evolved. The direction would be to find a suitable place for the documentation, where it would be available for all interested parties, and to develop search functionality.

Kasutatud kirjandus

1. Adzic, G. Specification By Example. Shelter Island, NY: Manning Publications Co.,2011.
2. Bali, M. Drools JBoss Rules 5.0: Developer's Guide. Chapter 5. Human-readable rules. Birmingham, UK : Packt Publishing, 2013.
3. Bennet, P., Coleman, G. Should I be using JBoss Rules, jBPM, SQL or just plain Java?, 2006 [WWW] https://community.jboss.org/wiki/JBossRulesjBPMSQLorJava (20.04.2014)
4. Boyer, J., Mili, H. Agile Business Rule Development: Process, Architecture and JRules Examples. Heidelberg: Springer, 2011.
5. Browne, P. Using Drools in Your Enterprise Java Application. — *O'Reilly*, 2005. [WWW] http://www.onjava.com/pub/a/onjava/2005/08/24/drools.html (20.04.2014)
6. Browne, P. JBoss Drools Business Rules. Birmingham, UK: Packt Publishing, 2009.
7. Business Rules for Better Operational Decisions [WWW] http://www-01.ibm.com/software/info/business-rules/ (20.04.2014)
8. Business Rule Management System. Hartmann Software Group, 2012. [WWW] http://www.hartmannsoftware.com/pub/Enterprise-Rule-Applications/brms (20.04.2014)
9. Concordion [WWW] http://www.concordion.org/ (20.04.2014)
10. Drools Documentation. Version 6.0.1.Final, 2013. [WWW] http://docs.jboss.org/drools/release/6.0.1.Final/drools-docs/html/index.html (20.04.2014)
11. Drools Documentation.Version 6.0.1.Final. 4.2. Build, Deploy, Utilize and Run, 2013. [WWW] http://docs.jboss.org/drools/release/6.0.1.Final/drools-docs/html/KIEChapter.html#BuildDeployUtilizeAndRunSection (20.04.2014)
12. Drools Documentation.Version 6.0.1.Final. 6.2.1. Agenda, 2013. [WWW] http://docs.jboss.org/drools/release/6.0.1.Final/drools-docs/html/ch06.html#d0e3790 (20.04.2014)
13. Drools Documentation.Version 6.0.1.Final. 7.8.3. Left Hand Side (when) syntax, 2013. [WWW] http://docs.jboss.org/drools/release/6.0.1.Final/drools-docs/html/DroolsLanguageReferenceChapter.html#d0e6903 (20.04.2014)

14. Drools Documentation.Version 6.0.1.Final. 7.8.4. The Right Hand Side (then), 2013. [WWW] http://docs.jboss.org/drools/release/6.0.1.Final/drools-docs/html/DroolsLanguageReferenceChapter.html#d0e9051 (20.04.2014)
15. Drools Documentation.Version 6.0.1.Final. 6.5.7. Rule Templates, 2013. [WWW] http://docs.jboss.org/drools/release/6.0.1.Final/drools-docs/html/ch06.html#d0e4969 (20.04.2014)
16. Drools Documentation.Version 6.0.1.Final. 7.8.5. Conditional named consequences, 2013. [WWW] http://docs.jboss.org/drools/release/6.0.1.Final/drools-docs/html/DroolsLanguageReferenceChapter.html#d0e9292 (20.04.2014)
17. Drools Documentation.Version 6.0.1.Final. 7.10 Domain Specific Language, 2013. [WWW] http://docs.jboss.org/drools/release/6.0.1.Final/drools-docs/html/DroolsLanguageReferenceChapter.html#d0e9394 (20.04.2014)
18. Hakizumwami, D. Building Enterprise Services with Drools Rule Engine. — *O'Reilly*, 2007 [WWW] http://www.onjava.com/pub/a/onjava/2007/01/17/building-enterprise-services-with-drools-rule-engine.html (20.04.2014)
19. Information Framework (SID). TMForum. [WWW] http://www.tmforum.org/InformationFramework/1684/Home.html#TRCSearch/LIST (20.04.2014)
20. Infotehnoloogia. Sõnastik. Osa 20: Süsteemiarendus: EVS-ISO/IEC 2382-20:1998. Tallinn: Standardiamet, 1998.
21. Infotehnoloogia. Sõnastik. Osa 33: Hüpermeedium ja multimeedium: EVS 2382-33:2003. Tallinn: Standardiamet, 2003.
22. Intelligent transport systems. System architecture. 'Use Case' pro-forma template: ISO/TR 25102:2008(en). Technical Committee ISO/TC 204 , 2008. [WWW] https://www.iso.org/obp/ui/#iso:std:iso:tr:25102:ed-1:v1:en:term:2.2 (20.04.2014)
23. Fowler, M. RulesEngine, 2009. [WWW] http://martinfowler.com/bliki/RulesEngine.html (20.04.2014)
24. Road vehicles. Automotive multimedia interface. Part 1: General technical overview: ISO 22902-1:2006(en). Technical Committee ISO/TC 22/SC 3, 2006. [WWW] https://www.iso.org/obp/ui/#iso:std:iso:22902:-1:ed-1:v1:en:term:3.1.70 (20.04.2014)

25. Systems and software engineering — Systems and software Quality Requirements and Evaluation (SQuaRE) — System and software quality models. 4.2 Product quality model. ISO/IEC 25010:2011(en). Technical Committee ISO/IEC JTC 1/SC 7, 2011[WWW] https://www.iso.org/obp/ui/#iso:std:iso-iec:25010:ed-1:v1:en (26.04.2014)
26. Tirelli, E. Cookbook: How to Test Rules using xUnit , 2011. [WWW] http://planet.jboss.org/post/cookbook_how_to_test_rules_using_xunit (20.04.2014)
27. The Process Executive. Glossary [WWW] http://processexecutive.com/blog/bpm-guide/glossary/ (01.05.2014)
28. What is Business Rules Management? [WWW] http://www-01.ibm.com/software/websphere/products/business-rule-management/whatis/(20.04.2014)
29. Wiseman, G. Real-World Rule Engines. — *InfoQ*, 2006. [WWW] http://www.infoq.com/articles/Rule-Engines(20.04.2014)

Lisa 1 – Reeglifailid

technologyRules.drl

```
package rules.offering.common.technologyRules;

import …

dialect "mvel"
no-loop true

// Which technologies do not work with each other
rule notActiveTechnology
    salience 1010
    agenda-group "init"
    when
        $t: Technology($doNotWorkWith : doNotWorkWith)
        $t2:Technology()
        Technology(name == $t2.name) from $doNotWorkWith
    then
        modify($t) { setActive( false ) };
end

// Which technology has the biggest priority
rule technologyPriority
    salience 1000
    agenda-group "init"
    when
        accumulate (Technology( $priority : priority, active == true);$max :
max($priority))
    then
        insert(new IntegerFact("TechnologyPriority",$max));
end

// Set offered technology
rule technologyOffered
    salience 900
    agenda-group "init"
    when
        $priority: IntegerFact(name == "TechnologyPriority")
        $t: Technology (priority == $priority.value)
    then
        modify($t) { setOffered( true ) };
end
```

internetSpeedRules.drl

```
package rules.offering.internet.internetSpeedRules;

import …

dialect "mvel"
no-loop true
```

```
// Only offerings suitable with the priority technology are offered
rule offeredInternetSpeedOfferings
    agenda-group "offer"
    when
        $priority: IntegerFact(name == "TechnologyPriority")
        $t: Technology(priority == $priority.value)
        $offering: ProductOffering (!offered, productCategory ==
ProductCategory.INTERNET_SPEED,
           $characteristic: productSpecCharacteristics["TECHNOLOGY"],
$characteristic.lov.containsKey($t.name))
        then
        $offering.setOffered(true);
end

// Internet speed default values
rule internetSpeedDefaults
    agenda-group "add"
    when
        $parent: BasketOffering(id in (98,99), childItems.isEmpty())
        accumulate (ProductOffering( $priority : priority, offered) from
$parent.childOfferings.values();$max : max($priority))
        $internetSpeedOffering: ProductOffering(priority == $max, offered) from
$parent.childOfferings.values()
        $b: Basket()
        $bs: BasketService()
    then
        $bs.addOfferingToBasket($b,$internetSpeedOffering,$parent.basketItemId, false)
end;
```

tvPlaceRules.drl

```
package rules.offering.tv.tvPlaceRules;

import …

dialect "mvel"
no-loop true

// TV is  offered, when technology is in ADSL2+, ADSL2++, AMI, KM2+TV:, PON, VDSL2+
rule tvOffered
    agenda-group "offer"
    salience 1000
    when
        Technology(offered, name in ("ADSL2+", "ADSL2++", "AMI", "KM2+TV", "PON",
"VDSL2+"))
        $offering: ProductOffering(!offered, id in (20) )
    then
        modify($offering) {setOffered( true )};
end

// number of TV places allowed (max is 8)

rule tvPlaces4
    agenda-group "offer"
    salience 900
```

```
    when
        ProductOffering(offered, id in (20))
        Technology(offered, name == "VDSL2+")
        $offering: ProductOffering(id == 200, !offered)
    then
        modify($offering) {setMaxNumberAllowed(4)};
end

rule tvPlaces2
    agenda-group "offer"
    salience 900
    when
        ProductOffering(offered, id in (20))
        Technology(offered, name =="ADSL2++")
        $offering: ProductOffering(id == 200, !offered)
    then
        modify($offering) {setMaxNumberAllowed(2)};
end

rule tvPlaces1
    agenda-group "offer"
    salience 900
    when
        ProductOffering(offered, id in (20))
        Technology(offered, name =="ADSL2+")
        $offering: ProductOffering(id == 200 ,!offered)
    then
        modify($offering) {setMaxNumberAllowed(1)};
end

rule tvPlaceOffered
    agenda-group "offer"
    salience 800
    when
        BasketOffering(id in (20), $childItems: childItems.values())
        accumulate ($o: BasketOffering(id == 200) from $childItems; $count : count($o))
        $offering: ProductOffering(id == 200, !offered, maxNumberAllowed > $count)
    then
        $offering.setOffered(true);
end

rule tvPlaceOfferedWhenTVisNotInBasket
    agenda-group "offer"
    salience 800
    when
        not (exists BasketOffering(id in (20)))
        $offering: ProductOffering(id == 200, !offered)
    then
        $offering.setOffered(true);
end
```

themePackRules.drl

```
package rules.offering.tv.themePackRules;

import …
```

```
dialect "mvel"
no-loop true

rule errorWhenViasatFilmAndSportIsAlreadySelected
    agenda-group "pre_add"
    salience 1000
    when
       $offering: ProductOffering(productSpecId in (35,36))
       $b: Basket($items: basketItemsAndChildItems.values())
       BasketOffering(productSpecId in (37)) from  $items
    then
       $b.addMessage("To add "+$offering.name+ " you must remove Viasat Film And Sport
theme pack from the basket first!");
end

rule addViasatFilmAndSportInstead
    agenda-group "pre_add"
    salience 900
    when
        ProductOffering(productSpecId in (35,36))
        $b: Basket($items: basketItemsAndChildItems.values())
        $bs: BasketService()
        $basketItem: BasketOffering(productSpecId in (35,36)) from  $items
    then
        $bs.removeItemFromBasket($b, $basketItem.basketItemId, $basketItem.parentItemId,
false);
        $bs.addOfferingToBasket($b, 211, $basketItem.parentItemId, false);
        $b.addMessage("Viasat Film and Sport was added instead");
end

rule removeViasatFilmViasatSport
    agenda-group "add"
    when
        BasketOffering(productSpecId == 37)
        $b: Basket($items: basketItemsAndChildItems.values())
        $bs: BasketService()
        $basketItem: BasketOffering(productSpecId in (35,36)) from  $items
    then
        $bs.removeItemFromBasket($b, $basketItem.basketItemId, $basketItem.parentItemId,
false);
End
```

deviceModelRules.drl

```
package rules.offering.device.deviceModelRules;

import …

dialect "mvel"
no-loop true

// General

rule highSpeeds
    agenda-group "device"
```

```
    when
        eval(true)
    then
        StringListFact high_speeds = new  StringListFact("highSpeeds");
        high_speeds.add("200M/200M");
        high_speeds.add("300M/300M");
        high_speeds.add("500M/500M");
        insert(high_speeds);
end


// Router compatibility

// Business Constraints

// only Inteno is recommended for 100M
   rule notRecommendedWith100M
       salience 100
       agenda-group "device"
       when
           DeviceContext(internetSpeed == "100M/100M")
           $model: DeviceModel(modelId != 3)
       then
           modify($model) { setCompatibility(Compatibility.NOT_RECOMMENDED) };
   end

// only Inteno and One Access are recommended when exists TV component
rule notRecommendedWithTV
    salience 90
    agenda-group "device"
    when
        DeviceContext(existsTvComponent == true)
        $model: DeviceModel(modelId not in (3,6))
    then
        modify($model) { setCompatibility(Compatibility.NOT_RECOMMENDED) };
end

// Technical constraints


// VDSL technology not compatible
rule vdslNotCompatible
    agenda-group "device"
    when
        DeviceContext(technology in ("VDSL"))
        $model: DeviceModel(modelId not in (1,3))
    then
        modify($model) { setCompatibility(Compatibility.NOT_COMPATIBLE) };
end

// High speed not compatible models
rule highSpeedNotCompatible
    agenda-group "device"
    when
        DeviceContext($speed :internetSpeed)
        StringListFact(name == "highSpeeds", stringList contains $speed)
        $model: DeviceModel(modelId != 3)
    then
```

```
        modify($model) { setCompatibility(Compatibility.NOT_COMPATIBLE) };
end

// When technology is SDSL, GSHDSL only OneAccess is compatible
rule sdslNotCompatible
    agenda-group "device"
    when
        DeviceContext(technology in ("SDSL", "GSHDSL"))
        $model: DeviceModel(modelId != 6)
    then
        modify($model) { setCompatibility(Compatibility.NOT_COMPATIBLE) };
end

// One Access is not compatible for XDSL or WIMAX technology
rule oneAccessNotCompatible
    agenda-group "device"
    when
        DeviceContext(technology in ("XDSL", "WIMAX"))
        $model: DeviceModel(modelId in (6))
    then
        modify($model) { setCompatibility(Compatibility.NOT_COMPATIBLE) };
end
```

deviceOfferingRules.drl

```
package rules.offering.device.deviceOfferingRules;

import …

dialect "mvel"
no-loop true

// Help rule for inserting additional context
rule calculateDeviceContext
   agenda-group "offer"
   salience 1000
   when
       $basket: Basket()
       $ds : DeviceService()
       exists DeviceModel(!offered, modelType == DeviceModelType.ROUTER)
       not (exists DeviceContext())
   then
       insert($ds.calculateDeviceContext($basket));
end

rule deviceNetworkContractModelCompatibility
    agenda-group "pre_acc"
    salience 1000
    when
        $ds : DeviceService()
    then
        insert(new
StringFact("networkDeviceCompatibility",$ds.checkNetworkDeviceCompatibility().toString())
);
```

```
        insert(new
StringFact("contractDeviceCompatibility",$ds.checkContractDeviceCompatibility().toString(
)));
end

rule executeDeviceRules
    agenda-group "offer"
    salience 900
    when
        DeviceContext()
    then
        drools.setFocus("device")
end




// Device Model (accept from One Access) is offered when it is compatible
rule modelOffered
    agenda-group "offer"
    salience 800
    when
        $offering: DeviceModel(compatibility == Compatibility.COMPATIBLE)
    then
        $offering.setOffered(true);
end

// One Access is offered, only when technology is  SDSL or GSHDSL
rule oneAccessOffered
    agenda-group "offer"
    salience 700
    when
        DeviceContext(technology not in ("SDSL", "GSHDSL"))
        $offering: DeviceModel(modelId == 6)
    then
        $offering.setOffered(false);
end

// Binding use is not offered with B2C offerings when ...
rule bindingUseNotOfferedWhenNeverOrdered
    agenda-group "offer"
    salience 600
    when
        $offering: ProductOffering (marketSegments contains MarketSegment.B2C, !offered,
termsType == TermsType.BINDING_USE)
    then
        $offering.setOffered(false);
end


// OneAccess lease is default router device offering when technology is SDSL
rule modelDefaultOneAccess
    agenda-group "add"
    activation-group "modelDefault"
    salience 200
    when
        $parent: BasketOffering(id == 40, childItems.isEmpty())
        Technology (name in ("SDSL", "GSHDSL"), offered)
```

```
        $modelOffering: ProductOffering(id == 412) from $parent.childOfferings.values()
        $b: Basket()
        $bs: BasketService()
        then
            $bs.addOfferingToBasket($b,$modelOffering,$parent.basketItemId, false)
end

// Inteno lease is default router device offering
rule modelDefaultInteno
    agenda-group "add"
    activation-group "modelDefault"
    salience 100
    when
        $parent: BasketOffering(id == 40, childItems.isEmpty())
        $modelOffering: ProductOffering(id == 400) from $parent.childOfferings.values()
        $b: Basket()
        $bs: BasketService()
        then
            $bs.addOfferingToBasket($b,$modelOffering,$parent.basketItemId, false)
end
```

deviceDeliveryMethodRules.drl

```
package rules.offering.device.deliveryMethodRules;

import …

// Delivery Method rules


// Technician is always allowed when order is made by Technician
rule channelTechnician
    salience 500
    agenda-group "pre_acc"
    when
        Channel(this == Channel.TECH)
        BasketOffering(productCategory == ProductCategory.DEVICE,  $characteristic :
offeringCharacteristics["DELIVERY_METHOD"] )
    then
        $characteristic.clearLov();
        $characteristic.getLov().put("TECHNICIAN", "Technician");
end

// Technician is allowed, when installation package is ordered. If channel is store, then
Store is also allowed
// (only for router, st-box and ip phone offerings)
rule installationPackageOrdered
    salience 400
    agenda-group "pre_acc"
    when
        $basket: Basket(feasibility in (Feasibility.AUTO))
        BasketOffering (id == 429)  from $basket.basketItemsAndChildItems.values()
        BasketOffering(productCategory == ProductCategory.DEVICE, id not in
(428),$characteristic : offeringCharacteristics["DELIVERY_METHOD"] )
        do[technician]
        Channel(this == Channel.STORE)
```

```
    then
        $characteristic.getLov().put("STORE", "Store");
    then [technician]
        $characteristic.clearLov();
        $characteristic.getLov().put("TECHNICIAN", "Technician");
end

// Technician is allowed, when feasibility check is Long or Outing.If channel is store,
then Store is also allowed
// (only for router, st-box and ip phone offerings)
rule feasibilityLongOuting
    salience 300
    agenda-group "pre_acc"
    when
        Basket(feasibility in (Feasibility.LONG, Feasibility.OUTING))
        BasketOffering(productCategory == ProductCategory.DEVICE, id not in
(428),$characteristic : offeringCharacteristics["DELIVERY_METHOD"])
        do[technician]
        Channel(this == Channel.STORE)
    then
        $characteristic.getLov().put("STORE", "Store");
    then [technician]
         $characteristic.clearLov();
         $characteristic.getLov().put("TECHNICIAN", "Technician");
end


// Courier and Store are allowed, when Technician is not from every Channel
// (only for router, st-box and ip phone offerings)
rule courierStoreAllowed
    salience 200
    agenda-group "pre_acc"
    when
        BasketOffering(productCategory == ProductCategory.DEVICE, id not in
(428),$characteristics : getOfferingCharacteristics().values())
        $characteristic : OfferingCharacteristic(characteristic.code ==
"DELIVERY_METHOD", !getLov().containsKey("TECHNICIAN")) from $characteristics
    then
        $characteristic.clearLov();
        $characteristic.getLov().put("STORE", "Store");
        $characteristic.getLov().put("COURIER", "Courier");
end

// Terminal should be always delivered only by technician
rule terminalDeliveryMethods
    agenda-group "pre_acc"
    when
       BasketOffering(id in (428),$characteristics :
getOfferingCharacteristics().values())
       $characteristic : OfferingCharacteristic(characteristic.code == "DELIVERY_METHOD",
!getLov().containsKey("TECHNICIAN")) from $characteristics
    then
       $characteristic.getLov().put("TECHNICIAN", "Technician");
end
```

installationPackageSelectionRules.drl

```
template header
networkDeviceCompatibility
contractDeviceCompatibility
selectInstallationPackage
message

package rules.offering.common.installationPackageSelectionRules;

import …

template "installationPackageSelection"

rule "installationPackageSelection_@{row.rowNumber}"
    agenda-group "pre_acc"
    when

        $basket: Basket()
        not (exists BasketOffering(id == 429))
        eval(@{selectInstallationPackage})
        StringFact(name == "networkDeviceCompatibility", value ==
"@{networkDeviceCompatibility}")
        StringFact(name == "contractDeviceCompatibility", value ==
"@{contractDeviceCompatibility}")
        $bs: BasketService()
    then
        $bs.addOfferingToBasket($basket, 429, null, false);
        $basket.addMessage("@{message}");

end


end template
```

Lisa 2 – Käivitatavad spetsifikatsioonid

Ärireeglite dokumentatsioon

Siia tuleb ärireeglite esileht!!!

Edasi

- Pakkumiste ärireeglid
- Arvelduskampaaniate ärireeglid

Results generated by **Concordion**
in 1 ms on 18-apr-2014 at 17:53:47 EEST

Ärireeglite dokumentatsioon >

Pakkumiste ärireeglid

Pakkumistega seotud reeglid

Edasi

- Internetipakkumiste ärireeglid
- Televisioonipakkumiste ärireeglid
- Seadmepakkumiste ärireeglid
- Paigalduspakettide reeglid

Results generated by **Concordion**
in 4 ms on 13-mai-2014 at 20:25:06 EEST

Ärireeglite dokumentatsioon > Pakkumiste ärireeglid >

Seadmepakkumiste ärireeglid

Seadmetega seotud reeglid

Edasi

- Seadmemudeli pakkumiste reeglid
- Seadmemudeli sobivuse reeglid
- Seadme tarneviisi reeglid

Results generated by **Concordion**
in 5 ms on 27-apr-2014 at 17:51:32 EEST

Seadmemudeli pakkumise reeglid

Seadmemudel on pakutav siis, kui see sobib.
Erandiks on OneAccess 20, mida pakutakse ainult SDSL, GHSDSL tehnoloogial, kuigi see toetab ka optika tehnoloogiaid.

Näited

- Äri (**B2B**) interneti tähtajalise kasutuse (**BINDING_USE**) pakkumisega **XDSL** tehnoloogial **ilma** televisioonita lubatud mudelid on:

Pakkumise modelId	Mudeli nimi
401	Inteno D301
405	Thompson TG789vn
409	Thompson TG784

- Äri (**B2B**) interneti üüri (**LEASE**) pakkumisega **SDSL** tehnoloogial **ilma** televisioonita lubatud mudelid on:

Pakkumise modelId	Mudeli nimi
412	OneAccess 20

- Äri (**B2B**) interneti üüri (**LEASE**) pakkumisega **KM2+TV** tehnoloogial **koos** televisiooniga lubatud mudelid on:

Pakkumise modelId	Mudeli nimi
400	Inteno D301

Edasi

- Erakliendi tähtajalise kasutuse lubamise reeglid

Tähtajalise kasutuse lubamise ärireeglid

Tähtajalise kasutusega seadet tohib vormistada siis, kui klient ei ole seda samal tootel varem saanud.

Näited:

1. Uuel ehitusel tuleb välja äri interneti Inteno D301 ruuteri tähtajalise kasutuse pakkumine **401**. OK
2. Uuel ehitusel tuleb välja äri nutiTV Motorola VIP1003 tähtajalise kasutuse pakkumine **414**. OK

Seadmemudeli sobivuse reeglid

Ruuteri sobivuse reeglid

Ruuteri tehniline sobivus võib sõltuda tehnoloogiast või interneti kiirusest.

- Wimax tehnoloogial sobivad mudelid:
  Inteno DG301, Thomson TG585, ST780, Thomson TG784, Thomson TG789vn
- XDSL tehnoloogial sobivad mudelid:
  Inteno DG301, Thomson TG585, ST780, Thomson TG784, Thomson TG789vn
- SDSL, GSHDSL tehnoloogial sobivad mudelid:
  One Access 20
- Optika tehnoloogial kiirusega kuni 100M sobivad mudelid:
  Inteno DG301, Thomson TG585, ST780, Thomson TG784, Thomson TG789vn, One Access 20
- VDSL tehnoloogial sobivad mudelid:
  Inteno DG301, Thomson TG789vn
- Optika tehnoloogial kiirusega rohkem kui 100M sobivad mudelid:
  Inteno DG301

Erakliendi toodete ärilised piirangud:

- Optika tehnoloogial on soovituslikud mudelid Inteno DG301 või Thomson TG789vn.

Ärikliendi toodete ärilised piirangud:

- Kui mudelid vanemad kui TG784, TG789vn,Inteno DG301 või One Access 20 on mittesoovituslikud.
- Kui on lahenduse koosseisus tellitud ka televisioon, siis soovituslik mudel on Inteno DG301.
- Kui kiiruseks on valitud 100M, siis soovituslik mudel on Inteno DG301.

Näited

Mudeli modelId	Mudeli nimi	Tehnoloogia	Interneti kiirus	TV olemas	Sobivus
3	Inteno D301	AMI	200M/200M	Jah	Sobib
1	Thompson TG789vn	PON	75M/75M	Ei	Sobib
1	Thompson TG789vn	KM2+TV	50M/50M	Jah	Mittesoovituslik
1	Thompson TG789vn	KM2+TV	100M/100M	Ei	Mittesoovituslik
1	Thompson TG789vn	KM2+TV	300M/300M	Jah	Ei sobi
1	Thompson TG789vn	VDSL	30M/10M	Ei	Sobib
2	Thompson TG784	VDSL	20M/10M	Ei	Ei sobi
6	One Access 20	SDSL	8M/8M	Ei	Sobib
6	One Access 20	GSHDSL	4M/4M	Jah	Sobib
3	Inteno D301	SDSL	8M/8M	Ei	Ei sobi
6	One Access 20	WIMAX	1M/256K	Ei	Ei sobi

Paigalduspaketi valimise reeglid

Süsteem peab valima paigalduspaketi automaatselt, kui ei ole kindel, kas kliendil olev seade sobib või mitte.

Näited

Võrguseadme sobivus	Lepingulise seadme sobivus	Paigalduspakett automatselt valitud
Sobib	Sobib	Ei
Sobib	Ei ole teada	Ei
Sobib	Ei sobi	Jah

Lisa 3 – JMeter koormustesti tulemused

Summary Report

Name: Summary Report

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors Successes | Configure

Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
HTTP Request	1000	42	4	234	40,61	0,00%	312,9/sec	1289,44	4219,9
TOTAL	1000	42	4	234	40,61	0,00%	312,9/sec	1289,44	4219,9

Graph Results

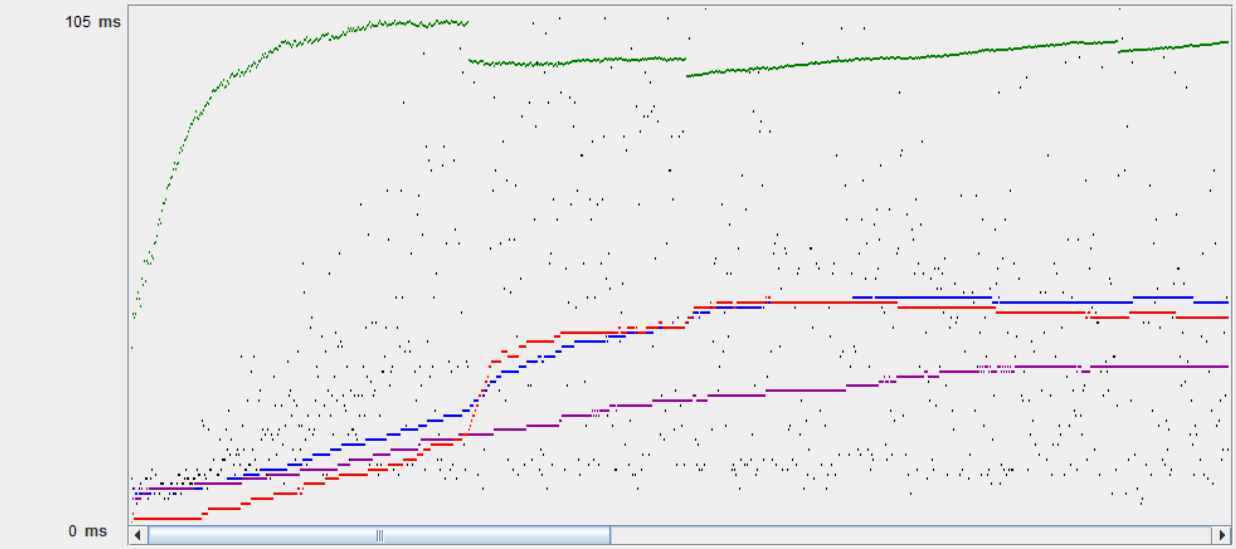
Name: Graph Results

Comments:

Write results to file / Read from file

Filename | Browse... | Log/Display Only: Errors Successes | Configure

Graphs to Display: Data, Average, Median, Deviation, Throughput



No of Samples 1000 | Latest Sample 5 | Average 42

Deviation 40 | Throughput 18 773,467/minute | Median 28