TALLINNA TEHNIKAÜLIKOOL
INFORMAATIKAINSTITUUT

Mudeliteisendusreeglite sünteesi ekspertsüsteem

Projekt õppeaines "IDX9022: Tehisintellekti rakendused info- ja teadmussüsteemides"

Koostaja: Mart Karu
091235IAQDD
karu@metal.ee

Juhendaja: Jaak Tepandi

TALLINN 2012

Sisukord

Sissejuhatus

Projekti ideeks on rakendada intelligentsete süsteemide tehnikaid mudelipõhise tarkvara-arenduses kasutatavate mudeliteisenduste reeglite genereerimiseks. Meetodi rakendamise näitena on vaatluse all kasutajaliideste abstraksetest mudelist kasutajaliidese lähtekoodi genereerimine.

Mudelipõhise arenduse ülevaade

Mudelipõhine arendus (Model Driven Development, MDD) on tarkvara-arendusmeetod, mis keskendub valdkonnaspetsiifilise teadmuse talletamisele mudelites, mida omakorda kasutatakse tarkvararakenduse automaatseks genereerimiseks või otse interpreteerivalt käivitamiseks. Diagrammil 1 on toodud kõikide MDD lähenemistele omased olulisemad mõisted, tegevused ja objektid ning nendevahelised seosed ja mõjud.

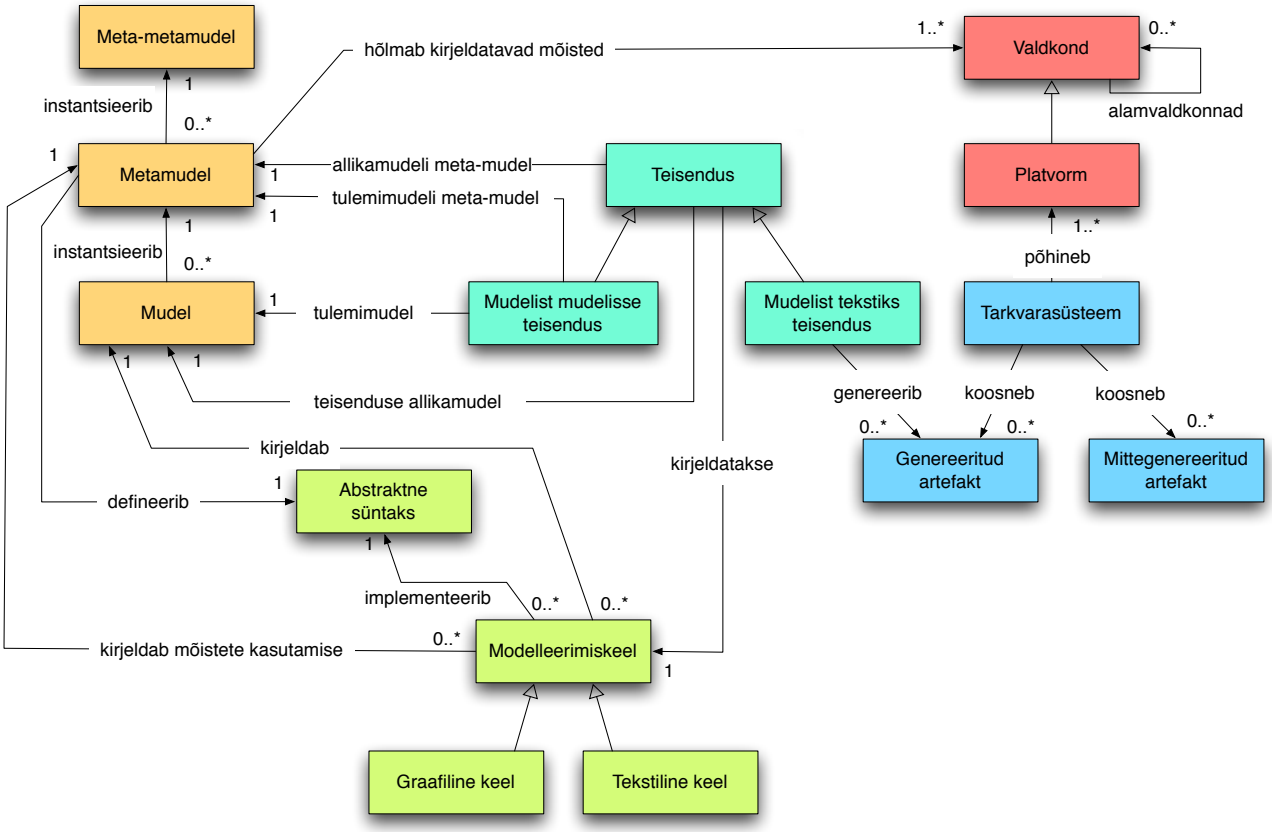


Diagramm 1: Mudelipõhise arendamise raamistik

MDD-s toimub mudelite kirjeldamine probleemvaldkonna tarbeks loodud modelleerimiskeelega, mille abil on võimalik kõige efektiivsemalt visualiseerida ning kirjeldada vaadeldavat valdkonda. Mudelites kasutatavad mõisted ja nende kasutamise reeglistiku kirjeldab mudeli metamudel, mis omakorda defineerib modelleerimiskeele jaoks keele loogilise struktuuri ehk abstraktse süntaksi. Mudelite metamudelite kirjeldamise viisi määrab meta-metamudel, mis kirjeldab metamudelite koostamiseks vajalikud mudelielemendid. Modelleerimiskeeled saavad olla graafilised või tekstilised.

Metamudelid hõlmavad endas valdkondade mõisteid. Valdkondadel võib olla alamvaldkondi ning üheks valdkondade hulgaks on tarkvaratehniliste platvormide valdkonnad, millel tugineb ka MDD tulemusena

tekkiv tarkvarasüsteem Mudeleid saab teisendada teistele mudelikujudele või tekstilisse esitusse, mis võib moodustada tarkvarasüsteemi genereeritud artefaktid. Mudelite teisendusreeglid, modelleerimiskeeled ning ka metamudelid kirjeldatakse samuti vastavate modelleerimiskeeltega.

Lähteülesanne

Tavapäraselt on teisendusreeglid koostatud ainult ühel kindlat viisil mudeliteisenduseks, kus algsest mudelist sünteesitakse teine mudel ühe kindla algoritmi järgi. Kui traditsioonilises mudeliteisenduses tekib teisendustes vajadus variatsioonideks, siis tuleb variatsioonid reeglitesse programmeerida tingimuslausetega või reeglite asendamisega. On ilmne, et mahukamate ja seetõttu keerukamate teisenduste juures muutuvad teisendusreeglid haldamisel keerukaks ning teisendusreeglitesse seguneb (lisaks mudeliteisendustele) ilmutamata kujul teadmust sihtplatvormi realiseerimisreeglite kohta.

Mudeliteisendusreeglite reeglisüsteemidepõhine genereerimine võimaldab teha mudeliteisendused paindlikuks seeläbi, et igakordseks mudeliteisenduseks valitakse vastavalt lähtemudeli iseloomule ning muudele piirangutele kõige sobilikumad teisendusreeglid. Kasuliku kõrvalefektina tekib taolise süsteemi kasutamiseks vajadus koostada modelleeritava valdkonna ning sihtplatvormide teadmust organiseeriv mustripank, mille alusel oleks võimalik teisendusreegleid koostada.

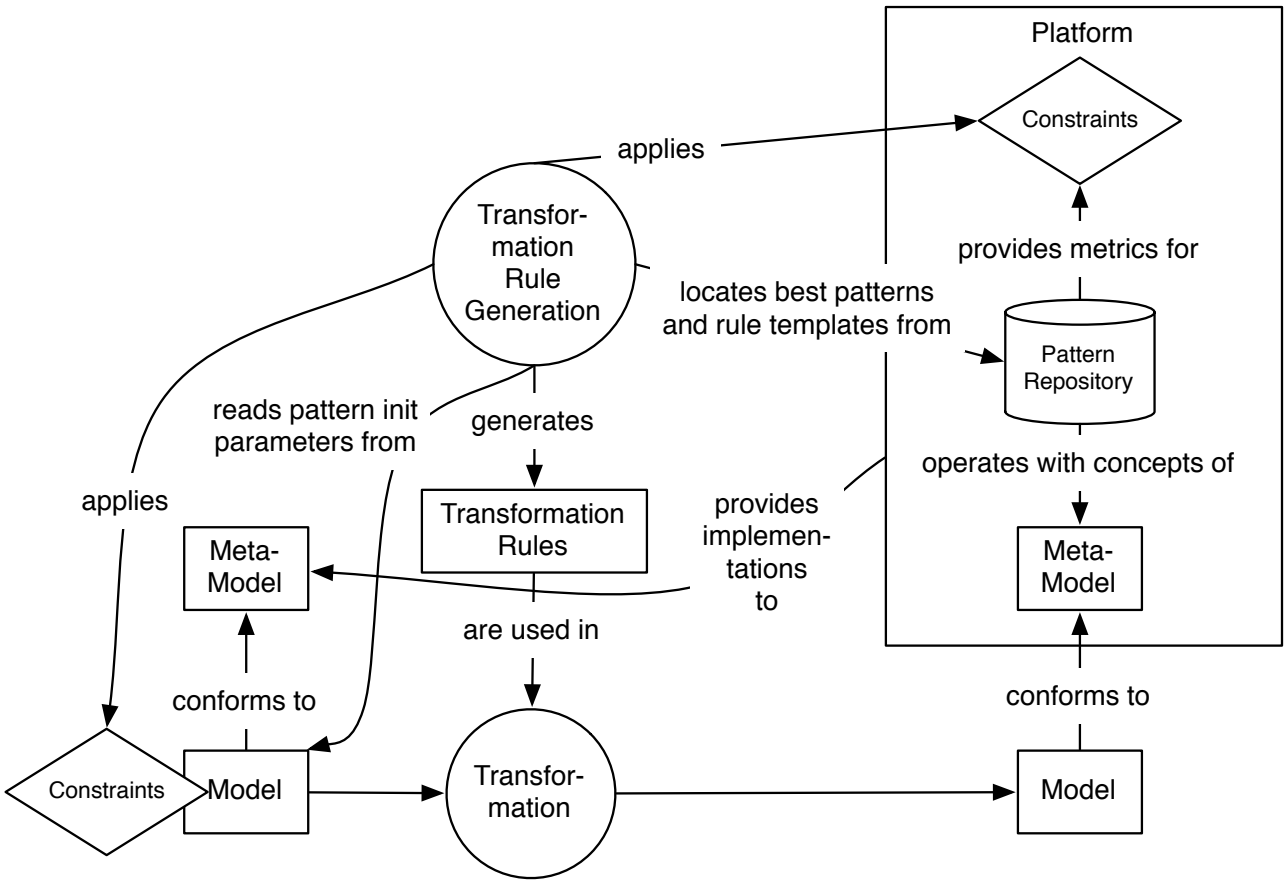
Mustripangas paiknev muster:

- kirjeldab sihtmudeli elementide võimalikku konfiguratsiooni, vastavalt sihtplavormi võimalustele, idioomidele ning headele tavadele.
- sisaldab teisendusreeglite malli või teisendusreeglite koostamise reeglit, mis defineerib selle, kuidas lähtemudelist koostada sihtmudelis mustrit realiseeriv konfiguratsioon.
- sisaldab mustrit iseloomustavaid atribuute, mille alusel on võimalik intelligentsete süsteemide tehnikatega valida teisndusteks kõige sobilikum muster.

Diagrammil 2 on kujutatud teisendurseeglite geneerimise üldine idee. Mustripank (Pattern Repository) koosneb sihtplatvormi mustritest ning reeglistikest, mille alusel saab valida vastavalt olukorrale asjakohaseimad mustrid. Mustrite valimisreeglid arvestavad sisendmudelis kirjeldatud info ja üldiste piirangute alusel parima mustri.

Näiteks kasutajaliideste modelleerimisel võib sisendmudelis olla viide abstraktsele interatksioonimustrile, näiteks kuupäeva valimisele andmisesestuses. Otsustuse tegemisel kasutatavateks piiranguteks võivad olla kasutajaliideste puhul näiteks eeldatava tarkvara kasutaja vilumus (professionaal vs algaja kasutaja) või tarkvara töökeskkonnaks oleva seadme ekraani suurus. Ekraani suurusest ja kasutaja vilumusest sõltub tarkvaras rakendatav kuupäeva sisestamise tarkvarakomponendi valik.

Parimate realiseerimismustrite valimise järel tuletatakse mustrist teisendusreeglid, mis rakendatakse algmudelile ja koostatakse sihtmudel, millest omakorda annab kirjeldatud meetodi järgi tuletada konkreestem mudel ja/või tarkvara lähtekood.



Diagamm 2: Teisendusreeglite genereerimine ja mustripank

Sisendid ja väljundid

Sisendiks on kasutaja kirjeldatud kasutajaliidese mudel ning piirangud. Projektis on ülesande lihtsustuse tõttu lähtemudeli metamudelis ainult üks element - DateEntryField, mille abil saab mudelis viidata sellele, et kasutajaliideses soovitakse kuupäeva sisestamise võimalust. Sisendmõistete üheks osaks olevas mudelis saavad olla ainult selle elemendi instantsid. Teiseks sisendmõistete osaks on piirangute komplekt, milleks on jällegi lihtsustuse mõttes projekti jaoks vaadeldud kahte kasutajaliideste elementide konkreetset realisatsiooni mõjutavat asjaolu - kasutajaliidese tööala suurust (ViewPortSize) ning kasutaja vilumust (UserSkillLevel). Projektis on mõõdetakse mõlema piirangu väärtuseid numbrite 0 ja 100 vahel.

Ülesande lihtsustamiseks ei ole vaatluse all nö liitmustreid (elemantaarsematest mustritest koostatud komponendid) ja kõrgema taseme kasutajaliidese mustreid (graafilise vaate paigutused, kõrgema taseme komponendid nagu menüüd jne).

Otsustuse lõplikuks väljundiks on mudelite teisendusreeglid tarkvara (või vahepealsete mudelite) genereerimiseks. Praktikas on piisavalt heaks väljundiks ka otsustuse käigus selgitatud sisendmudeli elemendile kõige asjakohasem realisatsioonimuster. Parimast mustrist on võimalik tuletada teisendusreeglid.

Väljundiks valitavad mustrid on HTML-põhiste kasutajaliideste genereerimiseks, ning mustrid on:

- kuupäeva sisestamise tekstikast

- mitu tekstikasti iga kuupäeva osa sisestamiseks
- valikud eeltäidetud sisuga
- ühte kuud kuvav kalender kuupäeva valimiseks
- ühe aasta kuid kuvav kalender kuupäeva valimiseks

Otsustussüsteemi oodatavaid valikuid etteantud sisendile ilmestab järgnev tabel:

		Kasutaja vilumus		
		Ekspert	**Keskmine**	**Algaja**
Tööala suurus	**Väike**	tekstiväli	valikud	valikud
	Keskmine	mitu tekstivälja	ühe kuu kalender	ühe kuu kalender
	Suur	mitu tekstivälja	ühe kuu kalender	aastakalender

Ekspert ootaks kiirema andmesisestuse huvides väiksema ekraaniala juures väiksemat tekstivälja ning suurema ala juures mitut tekstivälja. See võimaldaks hiirt või muud viitamisseadet vältida ning andmesisestuse saaks teha efektiivselt klaviatuurilt. Keskmise oskusega kasutaja ja algaja ootaksid segaduse vältimiseks eeltäidetud valikuid, Kui ekraanipind on suur ja kasutaja on algaja, siis võib näidata ka terve aasta ulatuses kuid kuupäeva valimiseks.

Võrdlus olemasolevate süsteemidega

Võrreldavaid süsteeme üldise meetodi osas ei ole leidunud. Kasutatud näite (kasutajaliideste genereerimine) osas on varasemalt koostatud mitmeid modelleerimis- ja teisendusmeetodeid, kuid nendes ei kasutata intelligentset mustrivalikut. Samuti on patenteeritud üldine mudel optimaalseimate kasutajaliidese elementide valimiseks, kuid seda ei käsitleta mudelipõhise arenduse raamistikus ning ei ole ka kirjeldatud valikumeetodi selgelt mõistetavat realisatsiooni[1].

Riskid ja otstarbekus

Ühtpidi võiks arvata, et sisend ja väljund on väga sarnased - mõlemad viitavad kasutajaliideste elementidele, kuid tegemist on siiski erinevate valdkondade mõistetega. Sisendmudel kirjeldab kasutajaliidese elemente abstraktsemas vormis, kus pole teada elemendi teostuse iseloomu sihtplatvormil. Väljundmõisted on samas kirjeldatud sihtplatvormi võimaluste kaudu. Kui näiteks ühe kasutajaliidese mudelis kirjeldatud kuupäeva sisestamise elemendi jaoks oleks graafilistes kasutajaliidestes võimalik kasutada komponente alates tavalisest tekstikastist kuni visuaalse kalendrini, siis graafilise liideseta dialoogipõhistes süsteemides (hääl-liidesed) tuleb see kasutajaliidese muster realiseerida küsimus-vastus dialoogiga (Süsteem: “Öelge palun kuupäev”; Kasutaja: “kümnes juuni kastuhat kaksteist”).

Põhimõtteliselt võiks mustrivaliku teostada programmeeritud valikualgoritmiga või otsustuspuuna, kuid valdkonna keerukuse ning kõige mustrite kombinatsioonide ja sisendite (mudel, piirangud) variatsioonide mahu tõttu ei ole see praktikas hallatav. Diagrammil 3 on toodud osade üldiste kasutajaliideste mustrite või komponentide loetelu ja nendevahelised seosed. On ilmne, et sellise keerukusega valdkonnas teadmuse salvestamine ja optimaalseima realisatsioonimustri valimine nõuab tõsisemat lahendust.

---

[1] http://www.google.com/patents/US20020118225

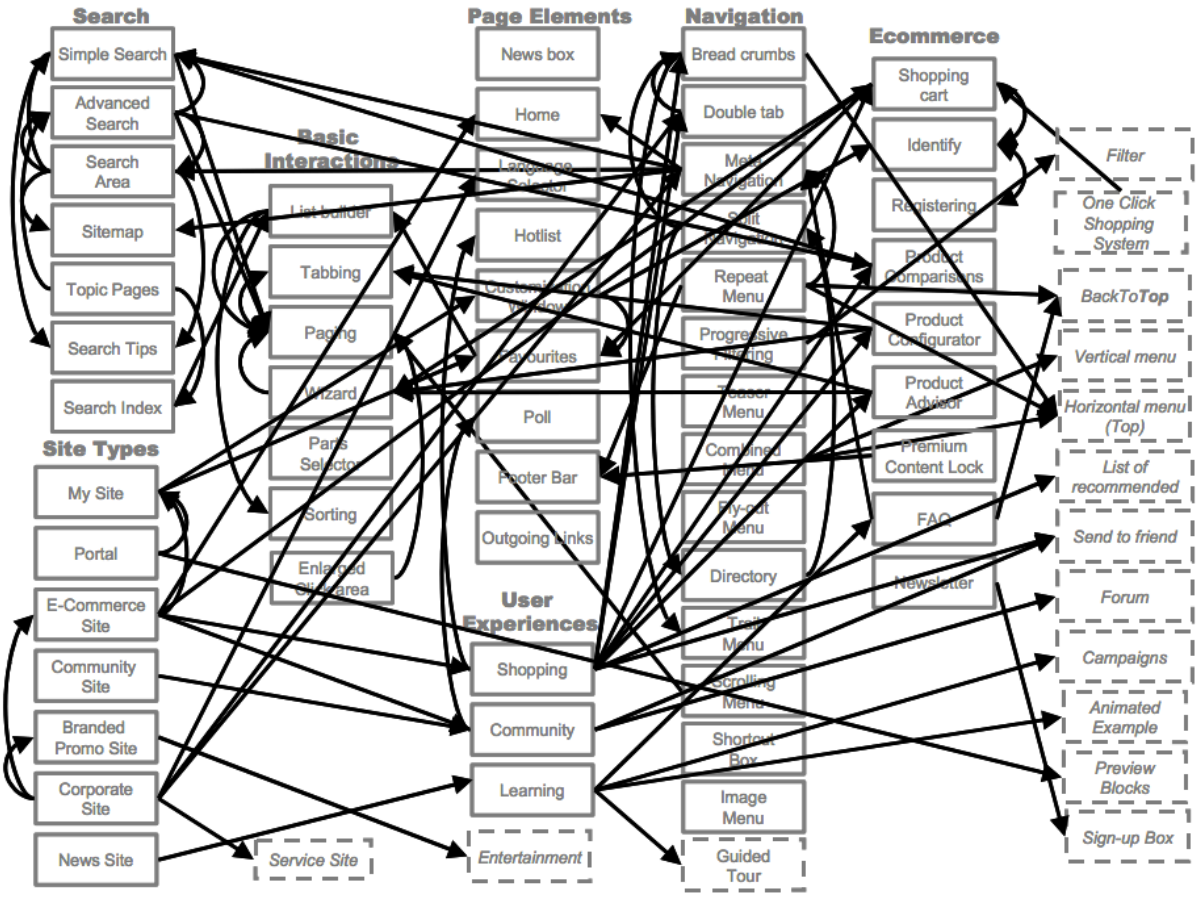


Diagramm 3: Kasutajaliideste üldiste mustrite seosed (laenatud artiklist "What makes a good User Interface pattern language?", E. Todd, E.Kemp, C.Phillips)

Realisatsioon

Platvorm

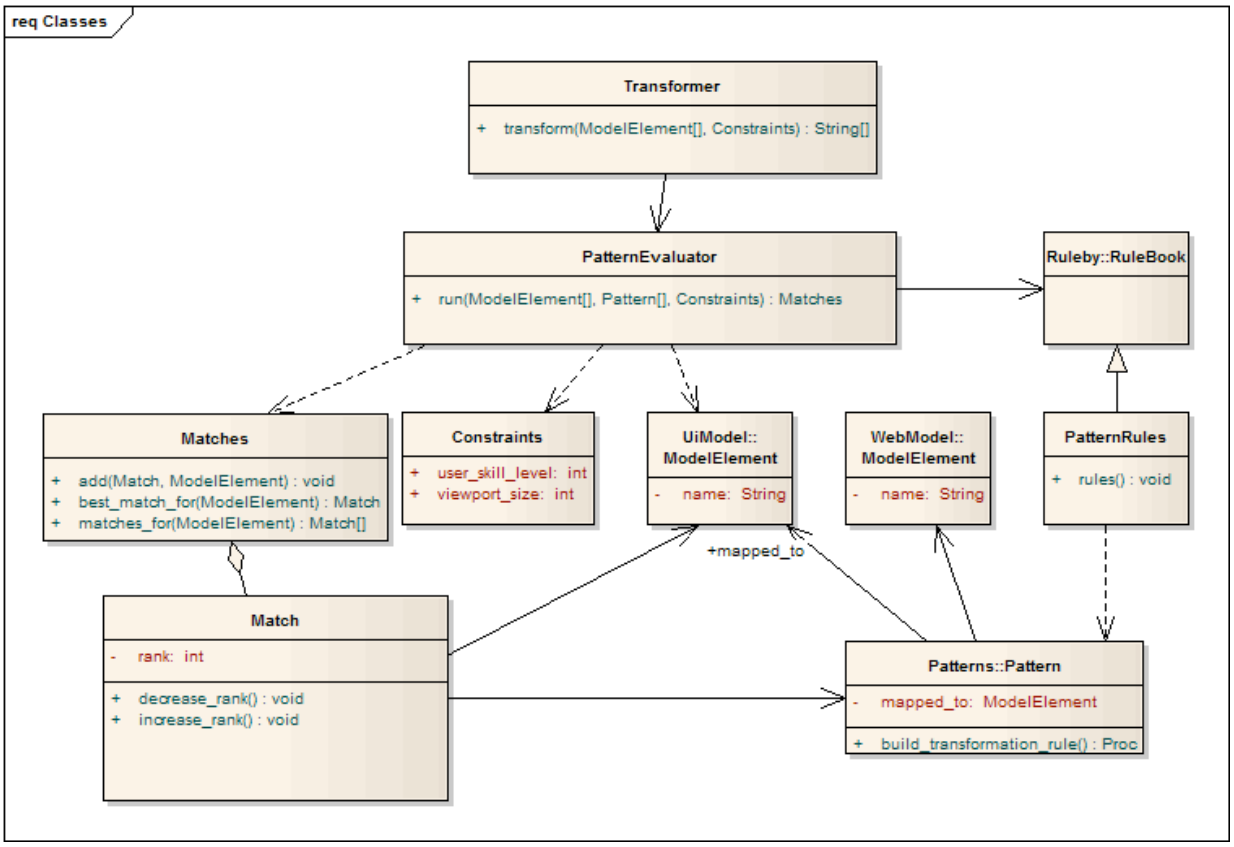
Kasutatud on keeles Ruby realiseeritud reeglitesüsteemi Ruleby, mida on laiendatud käesoleva projekti tarbeks reeglite vasaku tehtepoole (reeglite *match*) loogikatehete (meetodid `is?` ja `between?`) osas. Modifitseeritud Ruleby versioon asub aadressil https://github.com/martkaru/ruleby/tree/new_dsl.

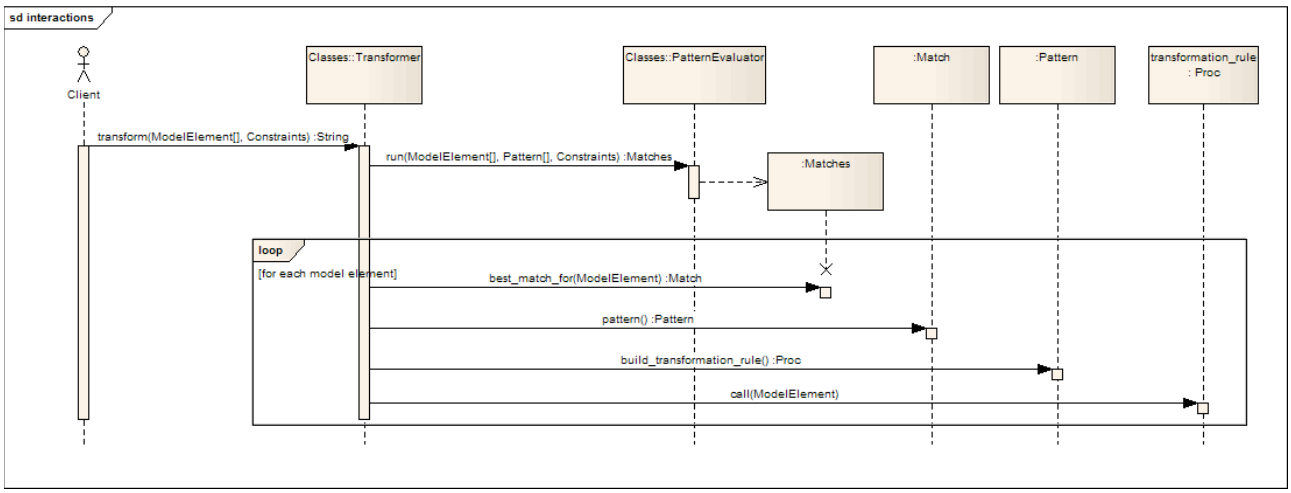
Mudelid ning teisendusreeglid on teostatud lihtsuse huvides objektorienteeritud võtetega keeles Ruby. Projekt pole keerukuse vältimiseks teostatud otseselt tavapäraste MDD vahenditega, vaatluse all on ainult mustrite reeglipõhise valimise loogika. Projekti on võimalik edasi arendada selliselt, et mudelid ning teisendusreeglid põhineks Ruby platvormile loodud MDD teekidel (RubyTL või RGen).

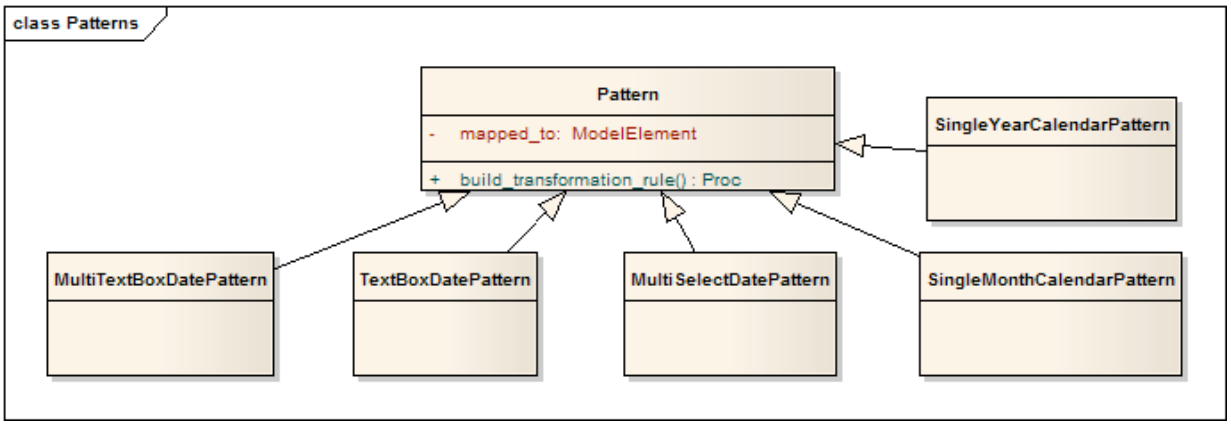
Realisatsiooni lähetkood asub aadressil https://github.com/martkaru/trans_rule_gen

Teostuse ülevaade

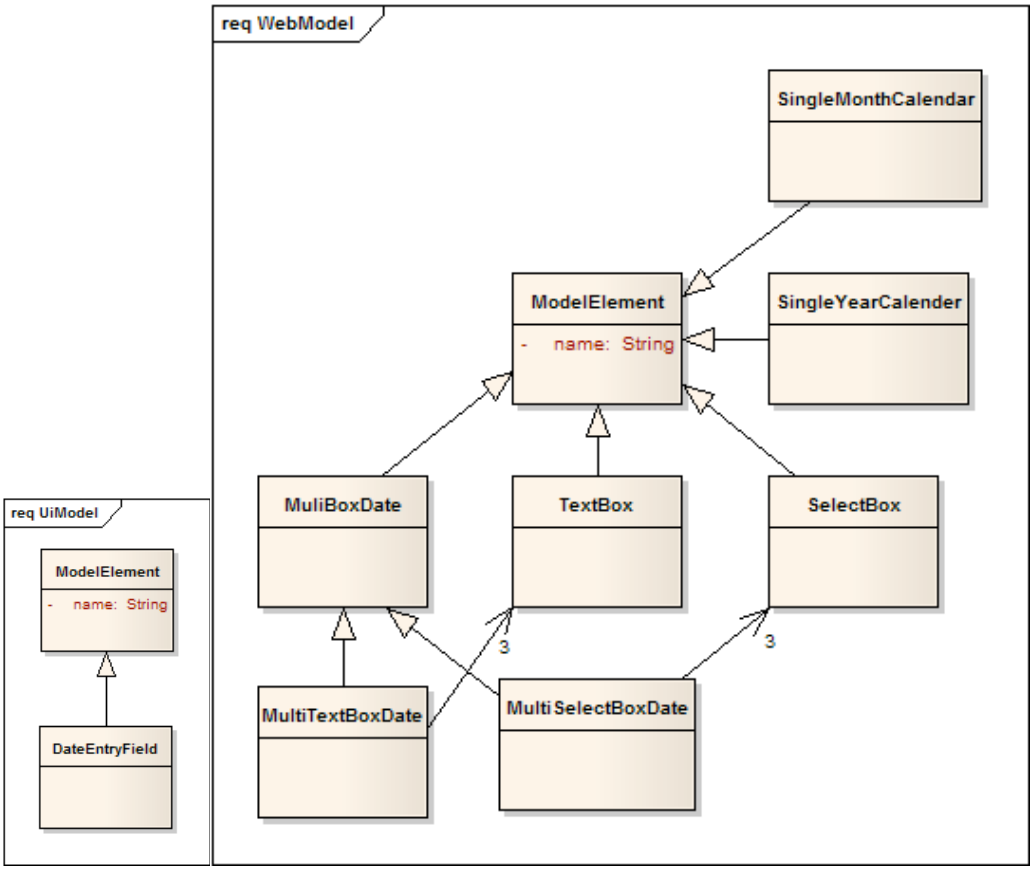
Alamsüsteemi sisendpunktiks on klassi Transformer meetod transform, mis teisendab etteantud mudeli ja piirangute järgi massiivi HTML liidese realisatsiooni tekstilõikudega. Transformer kasutab mustrite valimiseks klassi PatternEvaluator, mis initsialiseerib Ruleby reeglimootori ning sisestab sinna vajalikud objektid (faktid). Mustrivaliku reeglid on kirjeldatud klassis PatternRules, mis laiendab Ruleby klassi RuleBook. Muster (Pattern) kirjeldab sisendelemendi (UiModel::ModelElement) võimalikku teisendust sihtplatvormi elemendiks (WebModel::Element). Klass Match on vajalik mustrivaliku käigus sisendmudeli elementidele vastavate mustrite leidmiseks ja mustri sobilikkuse hindamiseks.







Sisendmudelis on ainult üks element (DateEntryField) ning sihtmudelis on lihtelemendid TextBox (HTML element <input type=”text” ...>), SelectBox (element <select ..>) ning nendele elementidele ehitatud liitmustrid MultiTextBoxDate ja MultiSelectBoxDate. SingleMonthCalendar ja SingleYearCalendar viitavad kalendrikuvamismustritele, mis on realiseeritud Javascript-põhiste komponentidega ning mille realiseerimiseks praeguses projektis piisab viitamisega vastava CSS stiiliklassile ning HTML DIV-elemendile (<div class=”calendar”).



Reeglisüsteemis toimub esmaselt vastavusseosed sisendmudeli elementide ja mustrite vahel, koostades klassi Match instantsid. Kuna sisendmudeli elemente on üks ning sellele elemendile vastavaid mustreid on mitu, siis on vastavussuhteid koostav reegel järgmine:

```
[TextBoxDatePattern, MultiTextBoxDatePattern, MultiSelectDatePattern,
      SingleMonthCalendarPattern, SingleYearCalendarPattern].each do |pattern_klass|
         rule [UiModel::DateEntryField, :m],
              [pattern_klass, :p],
              [Matches, :ms] do |v|
           assert(match = Match.new(v[:m], v[:p], 0))
           v[:ms].add(v[:m], match)
         end
      end
```

Iga mustri ja sisendmudeli elemendi kohta tehakse seos (Match) ning lisatakse see Matches objekti ning reeglisüsteemi faktiks.

Järgnevalt hinnatakse seoseid ning vastavalt piiranguparameetritele suurendatakse seose hinnangut (rank).

```
name :small_viewport_textbox
   rule [Constraints::Constraint, :c, where{ self.view_port_size.between?(0,29) }],
     [Match, :m, where{
       self.pattern.is?(TextBoxDatePattern)
       self.model_instance.is?(UiModel::DateEntryField)
   }] do |v|
     v[:m].increase_rank
   end
```

Testide sisendid

Automaattestid on teostatud BDD (Behaviour Driven Development) testimismeetodit toetava raamistikuga Rspec ning asuvad kataloogis spec.

Testide vaikimisi lähteandmed koostatakse järgnevate meetoditega. Mudel ehitatakse lihtsuse huvides tavalise massiivina:

```
def self.build_model
  elements = []
  elements << UiModel::DateEntryField.new(:name => 'from')
  elements << UiModel::DateEntryField.new(:name => 'to')
  elements
end
```

Mustripanga rolli täitvad sihtplatvormi kasutajaliideste mustrid luuakse samuti massiivina.

```
def self.build_patterns
  patterns = []
  patterns << TextBoxDatePattern.new(:for => UiModel::DateEntryField)
  patterns << MultiTextBoxDatePattern.new(:for => UiModel::DateEntryField)
  patterns << MultiSelectDatePattern.new(:for => UiModel::DateEntryField)
```

```
    patterns << SingleMonthCalendarPattern.new(:for => UiModel::DateEntryField)
    patterns << SingleYearCalendarPattern.new(:for => UiModel::DateEntryField)
    patterns
  end
```

Üldised sisendpiirangud, mis mõjutavad teisendusreeglite valikut luuakse vaikimisi kõige keskmistema väärtustega (parameetrite väärtused on vahemikus 0-100)

```
  def self.build_constraints
    Constraints::Constraint.new(
      :view_port_size => 50,
      :user_skill_level => 50
    )
  end
```

Testid

Testide käivitamiseks peab olema paigaldatud Ruby keskkond (versioon 1.9.3). Vajalike lisateekide paigaldamiseks tuleb projektikataloogis kasutada vahendit Bundler:

```
gem install bundler
bundle install
```

Reeglivaliku toimimist kirjeldav testis kontrollitaks kõiki sisendandmete kombinatsioone ning reeglisüsteemi vastuste asjakohasust. Iga testi juures muudetakse vajaduse korral vaikimisi sisendandmeid ja nende väärtusi. Test on koostatud Rspec testikeeles ning on üheaegselt suhteliselt hõlpsalt inimloetav ning automaat-testina käivitatav:

```
require 'spec_helper'

describe PatternEvaluator do
  subject { PatternEvaluator.new(PatternRules) }
  let(:model) { Factory.build_model }
  let(:patterns) { Factory.build_patterns }
  let(:constraints) { Factory.build_constraints }

  def run
    subject.run(model, patterns, constraints)
  end

  def best_match_for(e)
    run.best_match_for(e)
  end

  context "without constraints" do
    it "should generate no matches if no input is given" do
      subject.run([], [], []).matches.should be_empty
    end

    it "should generate no matches, if no patterns are given" do
      subject.run(model, [], []).matches.should be_empty
    end

    it "should generate no matches, if no model elements are given" do
      subject.run([], patterns, []).matches.should be_empty
    end

    it "should find match for each model element" do
      subject.run(model, patterns, []).matches.size.should == model.size
    end
```

```
end

context "with large viewport size" do
  before(:each) do
    constraints.view_port_size = 90
  end

  context "with expert user skill level" do
    before(:each) do
      constraints.user_skill_level = 90
    end
    specify { best_match_for(model.first).pattern.should be_kind_of(MultiTextBoxDatePattern) }
  end

  context "with average user skill level" do
    before(:each) do
      constraints.user_skill_level = 50
    end
    specify { best_match_for(model.first).pattern.should be_kind_of(SingleMonthCalendarPattern) }
  end

  context "with novice user skill level" do
    before(:each) do
      constraints.user_skill_level = 10
    end
    specify { best_match_for(model.first).pattern.should be_kind_of(SingleYearCalendarPattern) }
  end
end

context "with medium viewport size" do
  before(:each) do
    constraints.view_port_size = 50
  end

  context "with expert user skill level" do
    before(:each) do
      constraints.user_skill_level = 90
    end
    specify { best_match_for(model.first).pattern.should be_kind_of(MultiTextBoxDatePattern) }
  end

  context "with average user skill level" do
    before(:each) do
      constraints.user_skill_level = 50
    end
    specify { best_match_for(model.first).pattern.should be_kind_of(SingleMonthCalendarPattern) }
  end

  context "with novice user skill level" do
    before(:each) do
      constraints.user_skill_level = 10
    end
    specify { best_match_for(model.first).pattern.should be_kind_of(SingleMonthCalendarPattern) }
  end
end

context "with small viewport size" do
  before(:each) do
    constraints.view_port_size = 20
  end

  context "with expert user skill level" do
    before(:each) do
      constraints.user_skill_level = 90
    end
    specify { best_match_for(model.first).pattern.should be_kind_of(TextBoxDatePattern) }
```

```
    end

    context "with average user skill level" do
      before(:each) do
        constraints.user_skill_level = 50
      end
      specify { best_match_for(model.first).pattern.should be_kind_of(MultiSelectDatePattern) }
    end

    context "with novice user skill level" do
      before(:each) do
        constraints.user_skill_level = 10
      end
      specify { best_match_for(model.first).pattern.should be_kind_of(MultiSelectDatePattern) }
    end
  end
end

describe Transformer do
  subject { Transformer.new }
  let(:model) { Factory.build_model }
  let(:constraints) { Factory.build_constraints }

  it "should generate a web ui model" do
    subject.transform(model, constraints).should_not be_empty
  end

  it "should generate a web ui model that can be transformed to html" do
    subject.transform(model, constraints).map(&:render).should == [
      "<div class=\"month_calendar\" data-name=\"from\">",
      "<div class=\"month_calendar\" data-name=\"to\">"
    ]
  end
end
```

Testide käivitamise tulemuseks on järgmine väljund:

```
[martkaru@karu:trans_rule_gen:master:fa482cf]$ bundle exec rspec -f d

Constraints::Constraint
  should initialize instance var

PatternEvaluator
  without constraints
    should generate no matches if no input is given
    should generate no matches, if no patterns are given
    should generate no matches, if no model elements are given
    should find match for each model element
  with large viewport size
    with expert user skill level
      should be a kind of MultiTextBoxDatePattern
    with average user skill level
      should be a kind of SingleMonthCalendarPattern
    with novice user skill level
      should be a kind of SingleYearCalendarPattern
  with medium viewport size
    with expert user skill level
      should be a kind of MultiTextBoxDatePattern
    with average user skill level
      should be a kind of SingleMonthCalendarPattern
    with novice user skill level
```

```
      should be a kind of SingleMonthCalendarPattern
  with small viewport size
    with expert user skill level
      should be a kind of TextBoxDatePattern
    with average user skill level
      should be a kind of MultiSelectDatePattern
    with novice user skill level
      should be a kind of MultiSelectDatePattern

Transformer
  should generate a web ui model
  should generate a web ui model that can be transformed to html
```

Hinnang

Reeglisüsteemi kasutamine mustrite valimiseks tundub olevat üks võimalikke mõistlikke lahendusi algsele probleemile. Projektis sai kasutatud piiratud hulka elemente meetodi hõlpsamaks testimiseks, mistõttu võiks sellises mahus ülesande puhul kasutada ka tavalist algoritmilist realisatsiooni, kuid kui sisendis (mudelis ja piirangutes) kasvatada lähteinformatsiooni mahtu ning sellest informatsioonist sõltub optimaalseim mustrivalik, siis poleks mustrivaliku tulemus enam nii algoritmiliselt ootuspärane. Sellises olukorras saaks taolistele reeglitele ja seoste hindamisele (ranking) üles ehitada optimaalseima mustrivaliku, milles materjaliseeruv teadmus oleks üheltpoolt hallatav, kuid teisalt piisavalt keerukas et seda sama oleks võimatu teostada ettemääratud algoritmiga.

Närvivõrkudel põhinev mustrivalik võiks olla ka mõeldav, kuid selle tarbeks peaks võrgu õpetamiseks olema teada mustrivalikul oodatavad tulemused. Üks võimalus selleks võiks kasutajaliideste näite puhul olla võrgu õpetamine tagasisidega vastavalt tarkvara lõppkasutajate hinnangutele. Üldises meetodis mõne teises olukorras või valdkonnas rakendades võib sellise tagasiside saamine samas olla raskendatud, kuna inimene või muu lõpptulemust tarbiv agent ei tarvitse olla teadlik sellest, mis võiks olla parim mustrivalik või selle tulemusena sünteesitud väljund.

Kokkuvõtvalt võib arvata, et reeglipõhist mustrivalikut võiks edasi uurida ning teadmuse talletamiseks vajalikke reeglite kirjeldamise võtteid edasi arendada. Kuna tööst jäi välja keerukamate liitmustrite ja piirangute kasutamine, siis oleks võimalusi jätkuvaks tööks küll.

Lisa: Reeglite kirjeldused

```
class PatternRules < Ruleby::Rulebook
  def rules
    name :match_input_elements

    [TextBoxDatePattern, MultiTextBoxDatePattern, MultiSelectDatePattern,
       SingleMonthCalendarPattern, SingleYearCalendarPattern].each do |pattern_klass|
          rule [UiModel::DateEntryField, :m],
               [pattern_klass, :p],
               [Matches, :ms] do |v|
            assert(match = Match.new(v[:m], v[:p], 0))
            v[:ms].add(v[:m], match)
          end
       end

    name :small_viewport_textbox
```

```
rule [Constraints::Constraint, :c, where{ self.view_port_size.between?(0,29) }],
  [Match, :m, where{
    self.pattern.is?(TextBoxDatePattern)
    self.model_instance.is?(UiModel::DateEntryField)
}] do |v|
  v[:m].increase_rank
end

name :small_viewport_multiselect
rule [Constraints::Constraint, :c, where{ self.view_port_size.between?(0,29) }],
   [Match, :m, where{
    self.pattern.is?(MultiSelectDatePattern)
    self.model_instance.is?(UiModel::DateEntryField)
}] do |v|
  v[:m].increase_rank
end

rule [Constraints::Constraint, :c, where{ self.view_port_size.between?(30,75) }],
   [Match, :m, where{
    self.pattern.is?(MultiTextBoxDatePattern)
    self.model_instance.is?(UiModel::DateEntryField)
}] do |v|
  v[:m].increase_rank
end

rule [Constraints::Constraint, :c, where{ self.view_port_size.between?(30,75) }],
   [Match, :m, where{
    self.pattern.is?(SingleMonthCalendarPattern)
    self.model_instance.is?(UiModel::DateEntryField)
}] do |v|
  v[:m].increase_rank
end

rule [Constraints::Constraint, :c, where{ self.view_port_size.between?(76,100) }],
   [Match, :m, where{
      self.pattern.is?(MultiTextBoxDatePattern)
      self.model_instance.is?(UiModel::DateEntryField)
}] do |v|
  v[:m].increase_rank
end

rule [Constraints::Constraint, :c, where{ self.view_port_size.between?(76,100) }],
   [Match, :m, where{
    self.pattern.is?(SingleMonthCalendarPattern)
    self.model_instance.is?(UiModel::DateEntryField)
}] do |v|
  v[:m].increase_rank
end

rule [Constraints::Constraint, :c, where{ self.view_port_size.between?(76,100) }],
   [Match, :m, where{
    self.pattern.is?(SingleYearCalendarPattern)
    self.model_instance.is?(UiModel::DateEntryField)
}] do |v|
  v[:m].increase_rank
end

rule [Constraints::Constraint, :c, where{ self.user_skill_level.between?(0,32) }],
  [Match, :m, where{
    self.pattern.is?(MultiSelectDatePattern)
    self.model_instance.is?(UiModel::DateEntryField)
}] do |v|
  v[:m].increase_rank
end

rule [Constraints::Constraint, :c, where{ self.user_skill_level.between?(0,32) }],
```

```
      [Match, :m, where{
        self.pattern.is?(SingleMonthCalendarPattern)
        self.model_instance.is?(UiModel::DateEntryField)
    }] do |v|
      v[:m].increase_rank
    end

    rule [Constraints::Constraint, :c, where{ self.user_skill_level.between?(0,32) }],
      [Match, :m, where{
        self.pattern.is?(SingleYearCalendarPattern)
        self.model_instance.is?(UiModel::DateEntryField)
    }] do |v|
      v[:m].increase_rank
    end

    rule [Constraints::Constraint, :c, where{ self.user_skill_level.between?(33,65) }],
      [Match, :m, where{
        self.pattern.is?(MultiSelectDatePattern)
        self.model_instance.is?(UiModel::DateEntryField)
    }] do |v|
      v[:m].increase_rank
    end

    rule [Constraints::Constraint, :c, where{ self.user_skill_level.between?(33,65) }],
      [Match, :m, where{
        self.pattern.is?(SingleMonthCalendarPattern)
        self.model_instance.is?(UiModel::DateEntryField)
    }] do |v|
      v[:m].increase_rank
    end

    rule [Constraints::Constraint, :c, where{ self.user_skill_level.between?(66,100) }],
      [Match, :m, where{
        self.pattern.is?(TextBoxDatePattern)
        self.model_instance.is?(UiModel::DateEntryField)
    }] do |v|
      v[:m].increase_rank
    end

    rule [Constraints::Constraint, :c, where{ self.user_skill_level.between?(66,100) }],
      [Match, :m, where{
        self.pattern.is?(MultiTextBoxDatePattern)
        self.model_instance.is?(UiModel::DateEntryField)
    }] do |v|
      v[:m].increase_rank
    end
  end
end
```