Juho-Petteri Liukkonen

Liiketoimintalogiikan toteuttaminen sääntömoottorin avulla

Metropolia Ammattikorkeakoulu

Insinööri (AMK)

Tieto- ja viestintätekniikka

Insinöörityö

04.05.2023

Tiivistelmä

Tekijä:	Juho-Petteri Liukkonen
Otsikko:	Liiketoimintalogiikan toteuttaminen sääntömoottorin avulla
Sivumäärä:	42 sivua + 4 liitettä
Aika:	04.05.2023
Tutkinto:	Insinööri (AMK)
Tutkinto-ohjelma:	Tieto- ja viestintätekniikka
Ammatillinen pääaine:	Ohjelmistotuotanto
Ohjaajat:	Johtava asiantuntija Riku Sarlin Lehtori Vesa Ollikainen

---

Liiketoimintaa harjoittavien organisaatioiden toimintaa ohjaavat erilaiset tavoitteet, rajoitteet ja ehdot. Näiden tekijöiden pohjalta organisaatio voi muodostaa liiketoimintaprosesseja, joita ohjataan erilaisten liiketoimintasääntöjen avulla.

Tietojärjestelmiä toteuttaessa sovelluksen toimintaa varten määritellään vastaavasti prosesseja ja sääntöjä, jotka kertovat, millä tavoin sovelluksen tulisi toimia. Näiden komponenttien teknistä toteutusta kutsutaan sovelluksen liiketoimintalogiikaksi.

Opinnäytetyön tavoitteena oli tutustua liiketoimintaprosessien ja liiketoimintasääntöjen käsitteisiin tietojärjestelmien näkökulmasta ja tutkia, millä tavoin nämä kääntyvät ohjelmiston liiketoimintalogiikaksi.

Opinnäytetyön aikana toteutettiin eläkkeensaajan asumistuen laskentaohjelma, jonka tehtävänä oli annetun tiedon perusteella selvittää hakijan oikeus eläkkeensaajan asumistukeen ja saatavan tuen määrä euroina.

Projektin keskeisenä näkökulmana oli vertailla eri toteutustapoja ja selvittää liiketoimintasääntöjen ja -prosessien hallintaan tarkoitetun järjestelmän etuja liiketoimintalogiikan toteuttamisessa. Vertailun mahdollistamiseksi sovellus toteutettiin kahdella tavalla, Drools-sääntömoottorin avulla sekä ilman sitä.

Avainsanat: liiketoimintasääntö, liiketoimintaprosessi, sääntömoottori, Drools

Abstract

Author:	Juho-Petteri Liukkonen
Title:	Implementing Business Logic with Business Rules Engine
Number of Pages:	42 pages + 4 appendices
Date:	4 May 2023
Degree:	Bachelor of Engineering
Degree Programme:	Information and Communication Technology
Professional Major:	Software Engineering
Supervisors:	Riku Sarlin, Leading IT Specialist
	Vesa Ollikainen, Senior Lecturer

The actions of any business organization are guided by its goals and different restrictions and conditions it is affected by. Based on these variables a business can form its business processes that are controlled by various business rules.

In software development, this same approach can be used to form the business logic of an application. Business rules can be used to define the way an operation should work, and by chaining these operations together they form a business process inside the application. The implementation of these components in software is called business logic.

The goal of the study was to explore the possibilities of this approach by examining how business processes and business rules are implemented in an application. This goal was accomplished by developing a calculation program of a Finnish social security benefit, which would use its business logic to determine whether the social benefit could be granted, and the correct amount of the granted benefit.

One of the key aspects of the project was to compare different ways of implementing business logic of software and find out the advantages of using a business rule management system to implement business rules and processes. To make this comparison possible, the business logic was implemented in two ways; with the help of Drools business rules engine and without it.

Keywords: business rule, business process, rules engine, Drools

Sisällys

Liitteet

Lyhenteet

BLIP *Business Logic Integration Platform*. Liiketoimintalogiikan toteuttamiseen tarkoitettu kehitystyökalujen kokonaisuus.

BPMN *Business Process Model and Notation*. Liiketoimintaprosessien mallintamiseen ja kuvailuun käytetty standardisoitu merkintätapa.

BRMS: *Business Rule Management System.* Liiketoimintalogiikan toteutukseen ja hallintaan suunniteltu ohjelmisto, joka sisältää sääntömoottorin.

LCDP: *Low-Code Development Platform*. Ohjelmistojen kehittämiseen käytetty kehitysympäristö, jossa kehitystyötä tehdään pääsääntöisesti graafisen käyttöliittymän kautta, joka vähentää tarvetta kirjoittaa ohjelmakoodia.

POJO: *Plain Old Java Object*. Yksinkertainen Java-luokka, jota käytetään pääasiassa tiedon varastointiin.

REST: *Representational State Transfer*. Arkkitehtuurimalli ohjelmointirajapintojen rakentamiseen, joilla voidaan lähettää ja vastaanottaa tietoa HTTP- tai HTTPS-protokollan avulla.

UML: *Unified Modeling Language*. Standardoitu graafinen mallinnuskieli.

1 Johdanto

Jokaisen liiketoimintaa harjoittavan organisaation toimintaa ohjaavat erilaiset tekijät kuten sen tavoitteet ja toimintatavat sekä siihen kohdistuvat rajoitteet ja ehdot. Nämä tekijät huomioiden organisaatio voi muodostaa liiketoimintaprosesseja eli toimintojen ketjuja, joilla organisaatio pyrkii johonkin tavoitteeseensa. Prosesseihin kuuluvia toimintoja ohjataan erilaisten sääntöjen, liiketoimintasääntöjen, avulla. Liiketoimintasäännöt toimivat yksittäisen toiminnon suuntaviivoina, ja ne määrittelevät, mikä toiminnon lopputuloksen tulisi olla kussakin tilanteessa, kun otetaan huomioon toimintoon vaikuttavat tekijät.

Vastaavanlaista liiketoiminnan jakamista osiin voidaan hyödyntää myös tietojärjestelmien eli ohjelmistojen toteutuksessa. Ohjelmistoissa liiketoimintaprosessit ja liiketoimintasäännöt sisältyvät tyypillisesti suoraan ohjelmakoodiin, jossa ne muodostavat yhdessä sovelluksen liiketoimintalogiikan. Kyseinen toteutustapa voi kuitenkin tuottaa haasteita sovelluksen kehittäjille, sillä liiketoimintalogiikan sotkeutuessa muuhun sovelluslogiikkaan prosessien ja sääntöjen ohjelmointi, valvonta ja muokkaaminen vaativat laajaa teknistä osaamista ja ajankäyttöä, joka voi johtaa kehityksen hidastumiseen ja teknisen velan kasvuun.

Ratkaisuksi tähän ongelmaan on esitetty Low-Code-ratkaisuja käyttäviä BRMS-järjestelmiä, joiden avulla liiketoimintalogiikka voidaan eriyttää muusta sovelluslogiikasta. Lisäksi sääntömoottoreiden tuottajat esittävät, että Low-Code-ajattelutavan mukaisesti prosesseja ja sääntöjä voidaan lisätä, muokata, poistaa ja valvoa ilman merkittävää teknistä osaamista. Tällöin liiketoimintaprosessien ja -sääntöjen valvonta ja mahdollisuuksien mukaan jopa niiden implementointi sovellukseen olisi esimerkiksi liiketoiminnan asiantuntijan tehtävissä.

Opinnäytetyön aikana toteutetaan eläkkeensaajan asumistuen laskentaa suorittava sovellus sääntömoottorin avulla ja ilman sitä. Sovelluksen avulla pyritään selvittämään, voidaanko etuus myöntää syötettyjen tietojen perusteella ja kuinka suuri myönnettävän etuuden määrä on. Työn tavoitteena on kartoittaa eroavaisuuksia liiketoimintalogiikan toteutustapojen osaamis- ja laitteistovaatimuksissa, vertailla vaihtoehtojen suorituskykyä ja pohtia, millä tavoin sääntömoottorin käyttöönotto vaikuttaisi henkilöstön tehtävänjakoon ja työtaakkaan.

2 Liiketoimintasäännöistä sovelluksen liiketoimintalogiikaksi

Liiketoiminnan dokumentointi lähtee yleensä liikkeelle liiketoimintasuunnitelman laatimisesta, jossa määritellään organisaation liiketoiminnan perusajatus: mitä organisaatio tekee, kenelle se tekee, miksi ja miten (Liiketoimintasuunnitelma, 2019). Suunnitelmassa pyritään myös ottamaan huomioon toimintaan vaikuttavia tekijöitä kuten ehtoja, rajoitteita ja riskejä. Suunnitelmien pohjalta organisaatio voi luoda liiketoimintaprosesseja, jotka ovat erilaisten toimintojen sarjoja, joilla pyritään johonkin tavoitteeseen. Prosesseihin kuuluvia toimintoja kutsutaan liiketoimintaoperaatioiksi.

Yksittäisiä liiketoimintaoperaatioita voidaan ohjata liiketoimintasäännöillä. Liiketoimintasääntöjen (engl. Business Rules) tarkoituksena on määrittää, millä tavalla liiketoimintaoperaation kuuluisi toimia kussakin tilanteessa, kun otetaan huomioon siihen vaikuttavat tekijät. (What Are Business Rules?, n.d.)

Liiketoimintaprosessit, niiden sisällä olevat liiketoimintaoperaatiot ja operaatioita ohjaavat liiketoimintasäännöt ovat aina organisaation toiminnan taustalla, mutta riippuen yrityksen toimintatavoista niiden kuvaaminen ja dokumentointi voi vaihdella hyvinkin paljon. Esimerkiksi pankista lainaa hakevan henkilön luottokelpoisuuden tarkistamiseen liittyy erittäin tarkasti kuvattuja prosesseja ja sääntöjä, joita voidaan hyödyntää niin asiantuntijoiden kuin tietojärjestelmienkin toimesta, kun taas yhden henkilön pyörittämän saneerausyrityksen kohdalla liiketoimintaoperaatioihin – esimerkiksi omakotitalon kylpyhuoneen

laatoitukseen – liittyvät toiminnot, kuten laattojen ja muiden materiaalien määrän arviointi ja tilaaminen voivat olla suusanallisia tai täysin intuitiivisia. Siten niitä ei ole välttämättä dokumentoitu lainkaan.

Myös tietojärjestelmien eli ohjelmistojen suunnittelussa ja toteutuksessa voidaan hyödyntää liiketoiminnan jakamista prosesseiksi ja säännöiksi. Olennaista on tarkastella, millä tavoin nämä kääntyvät tietojärjestelmän liiketoimintalogiikaksi. Liiketoimintalogiikka (engl. Business Logic) tarkoittaa sovelluksen sitä osaa, johon liiketoimintasäännöt ja liiketoimintaprosessit on implementoitu (Frankenfield, 2020 ja Morris, 2016). Liiketoimintalogiikka on siis liiketoimintaprosessien, liiketoimintaoperaatioiden ja liiketoimintasääntöjen tekninen toteutus.

Ohjelmiston liiketoimintaprosessi voi olla esimerkiksi arvonlisäveron osuuden lisääminen jokaiselle laskun tuotteelle ja päivittämällä laskun loppusumma. Tähän prosessiin liittyy erilaisia operaatioita – kuten laskun tuotteiden hakeminen, yksittäisen tuotteen hinnan päivitys ja lopuksi laskun summan päivitys – joihin liittyy erilaisia liiketoimintasääntöjä, kuten tuotteen arvonlisäprosenttiluokka ja sitä tietyllä ajanhetkellä vastaavan arvonlisäveroprosentin hakeminen.

Alaluvuissa käydään läpi liiketoimintalogiikan komponenttien kuvaamista, määrittelyä ja toteuttamista edellä mainitun arvonlisäveroon liittyvän laskentaesimerkin avulla.

2.1 Liiketoimintaprosessit

Liiketoimintaprosessit (engl. Business Processes) ovat liiketoiminnallisten operaatioiden sarjoja, joilla pyritään johonkin tavoitteeseen (Gaikwad, n.d.). Liiketoimintaprosessit ilmenevät tietojärjestelmissä esimerkiksi seuraavasti: sovelluksen laskentaprosessi tarkoittaa sarjaan kytkettyjä funktiokutsuja, joiden päätteeksi yritetään päätyä johonkin tiettyyn lopputulokseen.

Tarkastellaan prosessin muodostamista käyttämällä esimerkkinä aiemmin mainittua laskentasovellusta, jossa laskulla olevien tuotteiden hintaan lisätään arvonlisäveron osuus. Yksinkertaisen laskentajärjestelmän käyttämään prosessiin voisivat kuulua seuraavat vaiheet:

1. tuotteen arvonlisäveroprosenttiluokituksen päättely tuotetyypin perusteella

2. tuotteelle asetettavan arvonlisäveroprosentin päättely edellisessä vaiheessa selvitetyn luokituksen perusteella

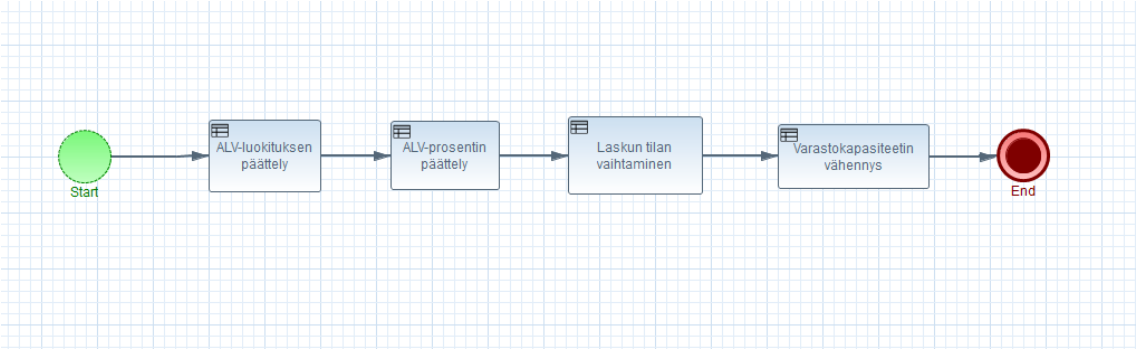
3. laskun tilan vaihtaminen valmiiksi

4. tuotteiden vähentäminen varastokapasiteetista

5. laskennan tulosten ilmoittaminen.

2.2 Liiketoimintaprosessin mallinnus

Liiketoimintaprosesseja voidaan mallintaa vuokaavioiden avulla (What Is Business Process Modeling?, 2021). Mallinnuksen tarkkuus ja tyyli voivat vaihdella riippuen siitä, onko kyseessä liiketoiminnan vai tietojärjestelmän suunnittelua varten tehty mallinnus. Prosessien mallinnukseen voidaan käyttää esimerkiksi BPMN-annotaatiota (*Business Process Model and Notation*) (kuva 1) tai UML-annotaatiota (*Unified Modeling Language)*, jotka ovat molemmat Object Management Groupin ylläpitämiä standardeja. Erona näillä annotaatioilla on lähinnä BPMN-annotaation erikoistuminen liiketoimintaprosessien mallinnukseen. BPMN-annotaation uusin versio on BPMN 2.0 eli lyhyemmin BPMN2.



Kuva 1. BPMN2-mallinnus laskentaprosessista.

Käytetään esimerkkinä luvussa 2.1 määriteltyä arvonlisäveron laskentaprosessia. Kuvan 1 kaaviossa on kuvattuna Drools-sääntömoottorin avulla toteutettu laskentasovellus. Prosessi lähtee liikkeelle alkutilanteesta, jossa sääntömoottorin työmuistiin on syötetty prosessissa käsiteltävät tiedot. Alkutilanteesta siirrytään ensimmäiseen toimintoon, joka on tässä tapauksessa ALV-luokituksen päättely. Sääntömoottori käy läpi operaatioon liittyvät liiketoimintasäännöt, eli tässä tapauksessa tarkistetaan, mikä ALV-luokitus kullekin laskun tuotteelle kuuluu. Prosessin seuraavat vaiheet toimivat samalla tavalla: sääntömoottori päättelee tuotteille arvonlisäveroprosentin edellisessä vaiheessa päätellyn luokituksen perusteella, jonka jälkeen tarkistetaan, että laskun tila voidaan vaihtaa VALMIS-tilaan. Tämän jälkeen tuotteiden kohdalta tehdään varastokapasiteetin vähennykset, ja laskun prosessointi on valmis.

2.3 Liiketoimintasäännöt

Liiketoimintasäännöt luovat pohjan sovelluksen liiketoimintalogiikan rakentamiselle. Säännöt pyritään muotoilemaan lausekkeiksi muodollisella ja loogisella kielellä, jotta ne olisivat yksiselitteisesti tulkittavissa sekä liiketoiminnan määrittelijöiden että teknisten toteuttajien puolesta. Ronald G. Ross ja Gladys S.W Lam luonnehtivat artikkelissa Structured Business Vocabulary: Concept Models liiketoimintasääntöjen selkeän sanoituksen tärkeyttä seuraavasti:

> ”That's the thing about business rules and other forms of formal business communications — they will be read by people you personally haven't met, in circumstances you might never have anticipated. Best to mean the words you say and to say exactly what those words mean.” (Ross & Lam 2016.)

Liiketoimintasääntöjen muotoilussa voidaan hyödyntää erikseen koottua liiketoimintasanastoa (engl. *Business Vocabulary)*, jolla pyritään yhdenmukaistamaan sääntöjen kieltä. Sanastossa määritellään kahdenlaisia komponentteja: termejä (engl. Terms) eli käsiteltäviä asioita ja sanoituksia (engl. Wordings) eli lauserakenteita, joilla pystytään ilmaisemaan näiden asioiden suhteita.

Yksittäinen liiketoimintasääntö on lauseke, joka määrittelee tapahtumaan liittyvän toimintatavan (Aliverti ym. 2016). Käytännössä liiketoimintasäännöt ovat monimuotoisia ehtolauseita. Sääntöjen muodostamiseen käytetään usein luonnollisen kielen vastineita logiikan kielen konnektiiveille kuten IF-ELSE, IF-THEN, ONLY-IF, WHEN-THEN ja niin edelleen. Yksinkertaisen liiketoimintasäännön rakenne voisi olla esimerkiksi seuraavanlainen:

```
WHEN <condition> AND <condition> THEN <action> ELSE <action>
```

Säännön rakenteen voidaan huomata muistuttavan jo valmiiksi ohjelmoinnissa käytettäviä ehtolauseita, ja ehtolauseista on pohjimmiltaan kysymyskin. Suomeksi käännettynä liiketoimintasääntö näyttäisi tältä:

```
Kun <ehtolauseke> ja <ehtolauseke> niin <seurauslauseke> ja muutoin
<seurauslauseke>
```

Säännössä on kaksi ehtolauseketta, joiden molempien tulee toteutua, jotta ensimmäinen seurauslauseke aktivoituu. Jos kumpikaan ehtolauseke ei toteudu tai edes toinen jää toteutumatta, aktivoidaan jälkimmäinen seurauslauseke.

Käydään läpi sääntöjen muodostusta aiemman laskentaesimerkin avulla. Päivittäistavarakaupassa käytössä olevan ohjelmiston tavoitteena on selvittää laskulle lisättyjen tuotteiden arvonlisäveroprosentti ja lisätä sen määrittämä osuus laskun loppusummaan. Järjestelmään luetaan elintarvike, esimerkiksi

leipäpussi, jonka hinta on 2,0 euroa. Kun tuote on lisätty laskulle, järjestelmä selvittää sen tuotetyypin perusteella tuotteelle kuuluvan arvonlisäveroprosentin. Suomessa elintarvikkeet kuuluvat alennettuun verokantaan, jonka arvonlisäveroprosentti on 14. Tämän tiedon perusteella mainitulle operaatiolle olisi mahdollista muodostaa ensimmäinen liiketoimintasääntö:

```
Jos <tuotteen tyyppi on elintarvike> niin <aseta tuotteen
arvonlisäveroprosentiksi 14>
```

Tämän jälkeen sovellus voi laskea tuotteen hinnan, johon on lisätty arvonlisäveron osuus, ja päivittää myös laskun loppusumman. Todellisuudessa säännöt eivät kuitenkaan välttämättä muodostu näin yksioikoisesti.

Jos tarkastellaan luvussa 2.1 esiteltyä prosessia, voidaan huomata, että näinkin yksinkertainen laskenta on jaettu moneen eri osaan, koska suunnittelussa on jouduttu ottamaan huomioon prosessiin vaikuttavia muuttujia. Liiketoimintasäännöissä voidaan käyttää esimerkiksi aikaan sidottuja muuttujia, jotka voivat ja luultavasti tulevatkin muuttumaan. Elintarvikkeet voidaan tulevaisuudessa luokitella johonkin toiseen verokantaan, verokantojen vastaavat veroprosentit voivat muuttua ja niin edelleen.

Laskentaprosessin ensimmäisessä operaatiossa selvitetään tuotteen verokanta seuraavan liiketoimintasäännön avulla:

```
Jos <tuotteen tyyppi on elintarvike> niin <aseta alennettu verokanta
1>
```

Prosessin seuraavassa vaiheessa päätellään tuotteen arvonlisäveroprosentin kerroin seuraavan säännön avulla:

```
Jos <tuote kuuluu alennettuun verokantaan (1)> niin <aseta tuotteen
veroprosenttikertoimeksi 0.14>
```

Prosessin seuraavissa vaiheissa havainnollistetaan laskun käsittelyyn liittyviä toimintoja, joista ensimmäinen on laskun tilan vaihtaminen Valmis-tilaan.

```
Jos <laskun tila on “avoin”> niin <vaihda laskun tilaksi “valmis”>
```

Laskun tila tarkistetaan prosessin viimeisessä vaiheessa eli varastokapasiteetin vähennyksessä. Tällä varmistetaan, että laskun käsittely on saatu valmiiksi ennen tuotteiden vähentämistä varastokapasiteetista.

```
Jos <laskun tila on “valmis”> ja <tuotteen tyyppi on “Leipäpussi”>
niin <vähennä tuotteen varastokapasiteettia yhdellä>
```

Liiketoimintaprosessi päättyy, ja seuraavaksi erillinen sovelluslogiikka päivittää laskun loppusumman, ja uusi hinta voidaan näyttää esimerkiksi käyttöliittymässä.

2.4 Liiketoimintasääntöjen implementointi tietojärjestelmässä

Kun prosessit ja niihin sisältyvät toiminnot sekä toimintoja ohjaavat liiketoimintasäännöt on suunniteltu, voidaan pohtia, millä tavalla ne saadaan muutettua tietojärjestelmän liiketoimintalogiikaksi. Liiketoimintasäännöt on perinteisesti toteutettu sisällyttämällä ne sovelluksen ohjelmakoodiin erilaisina algoritmeina ja ehtolauseina. Koodiesimerkissä 1 on esitetty tuotteen alv-kannan vaihtaminen sen tuotetyypin perusteella.

```java
package com.sample;

import data.Alvluokitus;
import data.Tuote;
import data.Tuotetyyppi;

public class ExampleApplication {

    public static void main(String[] args) {

        Tuote tuote = new Tuote();
        tuote.setNimi("Leipäpussi");
        tuote.setTuotetyyppi(Tuotetyyppi.ELINTARVIKE);
        tuote.setHinta(2.0);

        paatteleVerokanta(tuote);

    }

    public static void paatteleVerokanta(Tuote tuote) {
        if(tuote.getTuotetyyppi().equals(Tuotetyyppi.ELINTARVIKE)) {
            tuote.setAlvluokitus(Alvluokitus.ALENNETTU1);
        }
    }


}
```

Esimerkkikoodi 1. Liiketoimintasääntö sisällytettynä lähdekoodiin.

Tässä ”perinteisessä” toteutustavassa voidaan kuitenkin havaita useita haasteita. Liiketoimintasääntöjen implementointi vaatii ohjelmointikokemusta eli toisin sanoen teknistä osaamista. Lisäksi liiketoimintasäännöt ovat ohjelmakoodin seassa, jolloin myös sovellusarkkitehtuuriin joudutaan kiinnittämään entistä enemmän huomiota, jotta liiketoimintalogiikka ei sekoitu muuhun sovelluslogiikkaan.

Yksi merkittävimmistä haasteista liittyy kuitenkin liiketoimintasääntöjen valvontaan ja ylläpitoon. Perinteisellä ohjelmointitoteutuksella säännöt sisältyvät suoraan ohjelmakoodiin, jolloin niiden valvonta ja ylläpito vaativat enemmän aikaa ja teknistä osaamista. Luvussa 3 esiteltävät sääntömoottorit tarjoavat vaihtoehtoisen ja monipuolisemman tavan implementoida liiketoimintasääntöjä.

3 Sääntömoottorit

Sääntömoottorit ovat sovelluskehityksessä käytettyjä järjestelmiä, jotka tarjoavat vaihtoehtoisen tavan toteuttaa sovelluksen liiketoimintalogiikka. Näiden järjestelmien tavoitteena on helpottaa liiketoimintaprosessien ja liiketoimintasääntöjen hallintaa. Sääntömoottorit toimivat oman arkkitehtuurinsa varassa erillisenä komponenttina, joka voidaan kytkeä mihin tahansa sovellukseen, jossa ne vastaavat sovelluksen liiketoimintalogiikasta tai sen osasta.

Sääntömoottorit pyrkivät madaltamaan erottelua ihmisten kirjoittamien ja sovellusten tulkitsemien sääntöjen välillä, jolloin asiantuntijoiden laatimia sääntöjä voidaan käyttää liiketoimintalogiikan toteuttamisessa sellaisenaan. Sovellusarkkitehtuurin kannalta ero perinteiseen ohjelmointiin on selvä. Sääntömoottorin avulla liiketoimintasäännöt saadaan eristettyä ohjelmakoodista, jolloin sääntökokoelmaa voi ylläpitää liiketoimintaan perehtynyt asiantuntija ilman merkittävää teknistä osaamista, kun taas perinteisessä ratkaisussa toteutuksen tekee pätevöitynyt ohjelmoija. Sääntömoottori käy sääntöjä läpi ja tekee päätöksiä niiden perusteella. Tämä järjestelmä on puolestaan paketoitu erilleen varsinaisen sovelluksen rungosta omaksi kokonaisuudekseen. Lisäksi sääntömoottori voi toimia joko sovelluksen sisäisenä moduulina, tai se voidaan toteuttaa esimerkiksi erillisenä REST-rajapinnan tarjoavana mikropalveluna, mikä mahdollistaa entistä monipuolisemman sovellusarkkitehtuurin.

3.1 Low-code-alustat

Low-code-alustat ovat sovelluskehityksessä käytettyjä kehitysympäristöjä, joissa kehitystyö tapahtuu ohjelmakoodin kirjoittamisen sijasta graafisen käyttöliittymän kautta. Low-code-alustat pyrkivät vähentämään perinteisen ohjelmoinnin määrää automaation avulla ja tätä kautta nopeuttamaan ja helpottamaan uusien sovellusten kehittämistä, kun ohjelmakoodin kirjoittamiseen vaadittavaa teknistä osaamista ei juurikaan tarvita. Tällöin

perinteisesti ohjelmointiosaamista vaativia tehtäviä voidaan siirtää jonkin tietyn osa-alueen asiantuntijaryhmälle. Osaamisvaatimuksissa täytyy kuitenkin ottaa huomioon kehitettävän sovelluksen vaatimukset teknisen osaamisen suhteen. Nimensä mukaisesti Low-code-alustat tarjoavat kuitenkin mahdollisuuden kirjoittaa tarvittaessa osia liiketoimintalogiikasta perinteisen ohjelmoinnin avulla toisinkuin no-code-alustat, joissa ohjelmakoodin kirjoittamista ei vaadita välttämättä lainkaan. (What is low-code?, n.d.)

Esimerkki low-code-ratkaisuja tarjoavasta kehitysalustasta on opinnäytetyön toteutusvaiheessa käytetty sääntömoottori Drools, jonka avulla liiketoimintalogiikan toimintaa voidaan hallita ohjelmakoodista eristettyjen sääntötiedostojen avulla.

3.2 Drools

Drools on avoimen lähdekoodin liiketoimintasääntöjen hallintajärjestelmä (BRMS), jota kehittää Red Hat. Drools on rakennettu Java-kielellä, ja se voidaan suorittaa Java-virtuaalikoneella. Drools sisältää sääntömoottorin, josta käytetään nimeä Drools Expert. Kuten sääntömoottorin nimi implikoi, Drools voidaan toimintaperiaattensa vuoksi luokitella *asiantuntijajärjestelmäksi* (engl. Expert system) eli tietojärjestelmäksi, joka jäljittelee asiantuntijan päätöksentekoa. (Drools-dokumentaatio, versio 6.4.0.)

Drools on itsessään osa työkalujen kokonaisuutta, jota kutsutaan liiketoimintalogiikan integrointialustaksi (Business Logic Integration Platform, BLIP). Tähän kokonaisuuteen kuuluvat Drools Guvnor (liiketoimintasääntöjen hallintatyökalu), Drools Expert (sääntömoottori), jBPM (prosessien hallintatyökalu) ja Drools Fusion (tapahtumaprosessointi). (Fiorini & Gopalakrishnan 2015.)

Sääntömoottorin kanssa työskentely sisältää kaksi vaihetta (Martin, 2023):

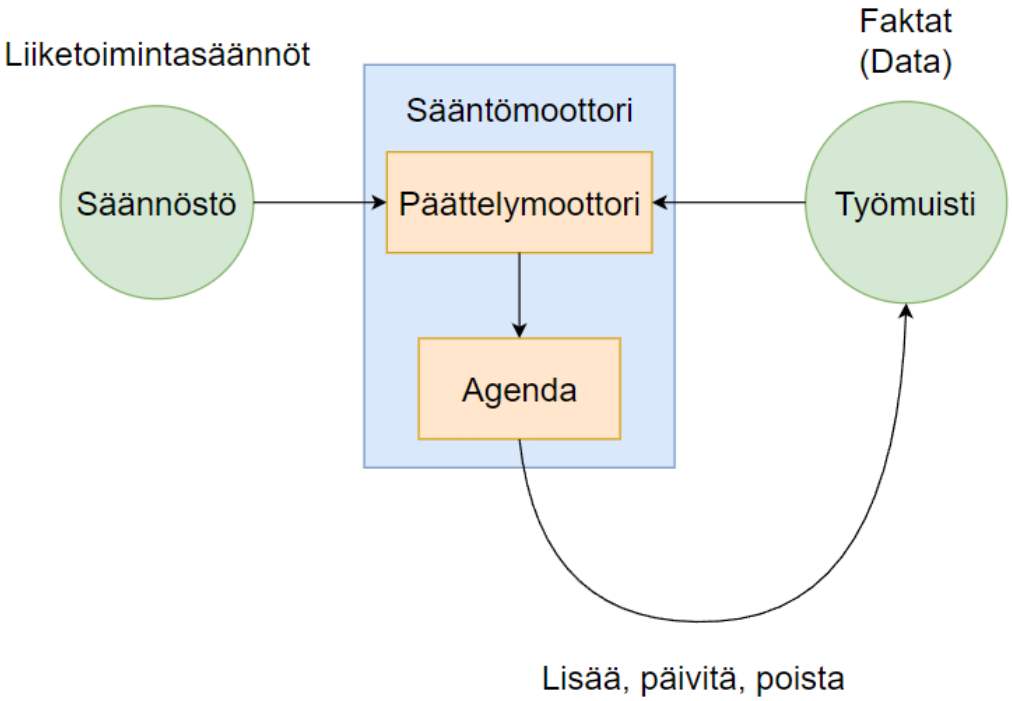
1. *Authoring* eli säännöstön laatiminen.

2. *Runtime* eli ohjelman ajaminen. Kun ohjelma ajetaan, sääntömoottorille syötetään dataa, jota sääntömoottori vertaa olemassa oleviin sääntöihin ja päätyy johonkin sääntöjen määrittämistä lopputuloksista.

Säännöstön laatimisvaiheessa säännöt suunnitellaan ja kirjoitetaan joko Droolsin omiin sääntötiedostoihin (.drl-tiedostot) tai Excel-tiedostoihin (.xls-tiedostot). Liiketoimintasäännöt implementoidaan sovellukseen ehtolauseina, joita sääntömoottori hallinnoi. Sääntömoottori tekee siis päätöksiä sille syötettyjen liiketoimintasääntöjen mukaisesti.

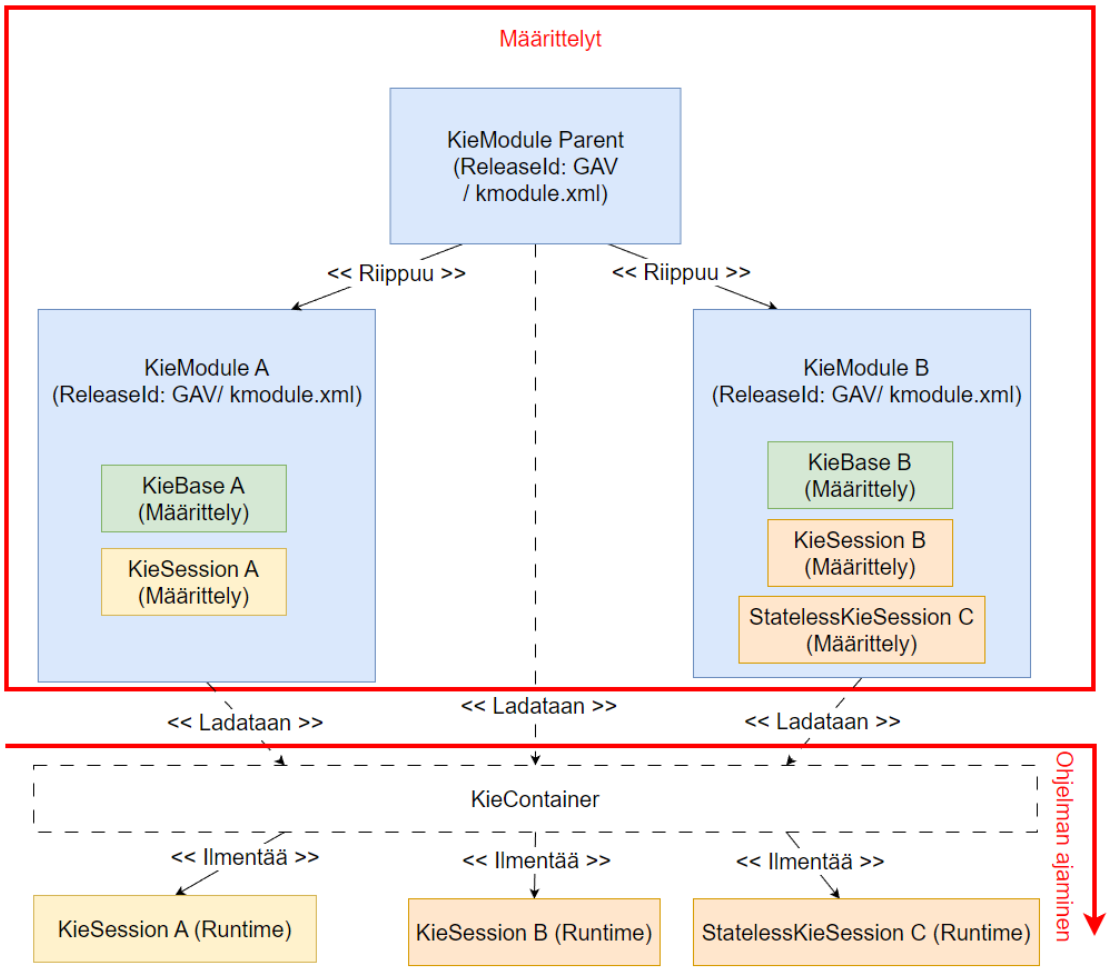
Logiikan ja datan erottelu on yksi sääntömoottorin tärkeimmistä ominaisuuksista ja eduista. Sääntömoottorin käyttämä data tulee ohjelmassa käytettävistä tietorakenteista ja logiikka puolestaan säännöistä. Näiden kahden osan muodostama kokonaisuus voidaan puolestaan erottaa muusta sovelluslogiikasta. Ohjelman suoritusvaiheessa sääntömoottorille syötetään dataa, joka on Droolsin tapauksessa yleensä yksinkertaisen Java-olion eli POJOn (*Plain Old Java Object*, yksinkertainen Java-luokka) muodossa. Droolsissa sääntömoottorille syötettyjä tietoja kutsutaan myös *faktoiksi*. Suorituksen aikana sääntömoottori vertaa sille syötettyä dataa olemassa oleviin sääntöihin. Kun säännössä oleva ehto täyttyy, suoritetaan säännössä määritelty toiminto. Säännöt määrittelevät sen, mitä tehdään eikä miten jokin asia tehdään. Kuvassa 2 on esitetty Droolsin arkkitehtuuri ylemmällä tasolla.

Päättelymoottorin (engl. Inference Engine) suorittamaa vertailua kutsutaan hahmontunnistukseksi (engl. Pattern Matching), ja siinä hyödynnetään tarkoitukseen optimisoitua algoritmia. Droolsin versiosta 6 lähtien käytössä on ollut PHREAK-algoritmi, joka syrjäytti aiemmin käytetyn ReteOO-algoritmin, joka puolestaan on olio-ohjelmointia varten optimisoitu muunnos Rete-algoritmista. Edeltäjäänsä verrattuna PHREAK skaalautuu tehokkaammin ja on nopeampi suurissa projekteissa.



Kuva 2. Droolsin arkkitehtuuri (Martin, 2023).

Ennen sääntömoottorin käyttöönottoa selvitetään ensin Droolsin eri komponenttien suhteet toisiinsa (kuva 3). Komponentteja ovat KieServices, KieContainer, KieModule, KieBase ja KieSession. Komponenttien nimissä käytetään KIE-etuliitettä (Knowledge Is Everything), joka viittaa yleisesti liiketoimintaan liittyvän tiedon käsittelyyn. KIE on myös kattotermi useille liiketoiminnan automatisointiin keskittyville avoimen lähdekoodin projekteille kuten Drools ja jBPM.



Kuva 3. Droolsin sisäinen rakenne. (Aliverti ym. 2016.)

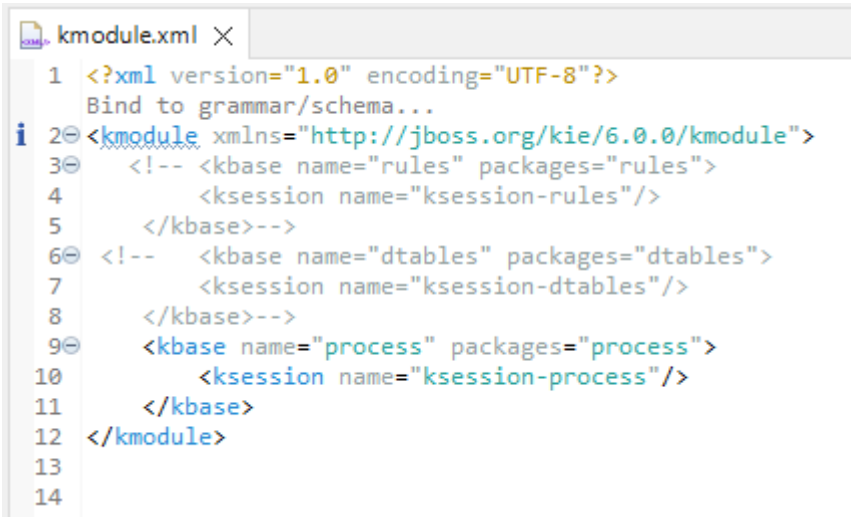
KieServices toimii järjestelmän pääkomponenttina, jonka avulla päästään käsiksi sääntömoottorin tarjoamiin palveluihin. KieServices-rajapinnasta voidaan hakea viittaus ainoastaan globaaliin singletoniin eli rajapinnan ainoaan toteuttavaan instanssiin. Viittaus KieServices-singletoniin haetaan seuraavalla tavalla:

```
KieServices ks = KieServices.Factory.get();
```

Haetun singletonin avulla voidaan luoda uusi KieContainer, joka toimii KieModulen suorittavana alustana.

```
KieContainer kContainer = ks.newKieClasspathContainer();
```

KieModule sisältää puolestaan ajon aikana vaadittavat resurssit kuten liiketoimintasäännöt ja prosessit (kuva 4).



Kuva 4. KieModulen sisältö.

Kuvassa 4 on esitetty KieModulen rakenne. KieBase tarkoittaa resursseista koottua suoritettavaa tietopakettia, ja yhdessä KieModulessa niitä voi olla useita. Kuvassa 4 kaksi muuta KieBase-määritystä on kommentoitu pois. KieBasen määrityksissä on avainsana *packages*, joka viittaa käytettävien resurssien sijaintiin hakemistossa.

Sääntömoottori suorittaa KieContaineriin ladattuja KieModuuleja, jotka sisältävät varsinaiset säännöt. Lopuksi KieContainerin avulla luodaan uusi KieSession:

```
KieSession kSession = kContainer.newKieSession("ksession-dtables");
```

KieSession puolestaan on sääntömoottorin instanssi, joka käyttää toiminnassaan KieBasessa olevia sääntötiedostoja. Droolsissa sessioita voi olla kahdenlaisia: tilattomia ja tilallisia. Stateless session eli tilaton sessio toimii kuten tavallinen funktio: sille syötetään dataa ja se palauttaa tulokset, eikä se muista mitään aiemmista suorituskerroista. Stateful session eli tilallisessa

sessiossa voidaan puolestaan tallentaa aiemman suorituskerran tulokset, ja se sallii näin iteratiiviset muutokset session olioihin. Tilattoman session käyttökohteita ovat esimerkiksi yksinkertaiset validoinnit ja laskutoimitukset. Tilallisia sessiota voidaan käyttää esimerkiksi osakemarkkinoiden valvontaan ja analysointiin.

Data syötetään sessioon insert()-metodilla ja sääntöjen hahmontunnistus voidaan aloittaa. Tilallisessa session metodi fireAllRules() käynnistää prosessin ja sääntömoottori käy vaiheittain läpi kaikki säännöt, jotka on suoritettavissa sääntömoottorin työmuistiin syötettyjen tietojen perusteella. Tilaton sessio käynnistetään puolestaan komennolla execute(). Kun säännön ehto täyttyy, säännössä kuvailtu toiminta aloitetaan. Tilallisen session jälkeen tulee kutsua dispose()-metodia, joka vapauttaa välimuistin ja ehkäisee tietovuotoja.

3.3 Liiketoimintasääntöjen implementointi Droolsilla

Liiketoimintasäännöt voidaan kirjoittaa joko Droolsin sääntötiedostoihin (.drl-tiedostot) tai taulukoihin (.xls- ja .csv-tiedostot). DRL-tiedostoissa sääntöjen kirjoittamiseen käytetään Droolsin omaa syntaksia, joka muistuttaa joiltain osin Java-ohjelmointikieltä (kuva 4).

```
package com.sample

import data.Tuote;
import data.Tuotetyyppi;
import data.AlvTiedot;

rule "Luokka 1: ALV-prosentti 24"
ruleflow-group "alvlaskenta"
    when
        tuote : Tuote( tyyppi == AlvTiedot.LUOKKA1)
    then
        tuote.setHinta(tuote.getHinta() + (tuote.getHinta() * AlvTiedot.LUOKKA1_KERROIN));
end

rule "Luokka 2: ALV-prosentti 14"
ruleflow-group "alvlaskenta"
    when
        tuote : Tuote( tyyppi == AlvTiedot.LUOKKA2)
    then
        tuote.setHinta(tuote.getHinta() + (tuote.getHinta() * AlvTiedot.LUOKKA2_KERROIN));
end

rule "Luokka 3: ALV-prosentti 10"
ruleflow-group "alvlaskenta"
    when
        tuote : Tuote( tyyppi == AlvTiedot.LUOKKA3)
    then
        tuote.setHinta(tuote.getHinta() + (tuote.getHinta() * AlvTiedot.LUOKKA3_KERROIN));
end

rule "Luokka 4: ALV-prosentti 0"
ruleflow-group "alvlaskenta"
    when
        tuote : Tuote( tyyppi == AlvTiedot.LUOKKA4)
    then
        tuote.setHinta(tuote.getHinta() + (tuote.getHinta() * AlvTiedot.LUOKKA4_KERROIN));
end
```

Kuva 5. DRL-sääntötiedoston sisältö.

Koska Drools on ohjelmoitu Java-kielellä ja sääntötiedostot käännetään Java-kielelle ennen ohjelman ajamista, DRL-sääntötiedostoista voidaan löytää samoja avainsanoja kuin Java-tiedostoista. Kuvassa 4 esitetyn sääntötiedoston ylälaidassa näkyy esimerkiksi *package*-avainsana, joka toimii samalla tavoin kuin Java-luokissa. Säännöt kuuluvat johonkin pakettiin, joka toimii niiden nimiavaruutena. Saman paketin sisällä olevien sääntöjen nimien tulee siis olla uniikkeja. Tiedoston ylälaidassa näkyy myös *import*-lausekkeet, jotka tuovat tarvittavat Java-luokat sääntöjen käyttöön.

Näiden lisäksi sääntötiedostoissa voidaan määritellä globaaleja muuttujia, funktioita, kyselyitä sekä tietenkin itse liiketoimintasääntöjä. Globaalit muuttujat tarkoittavat muuttujia, joihin voi viitata missä tahansa prosessin vaiheessa.

Globaalit muuttujat voivat olla esimerkiksi sääntötiedoston ulkopuolella määriteltyjen apuluokkien ilmentymiä, jotka sisältävät metodeja, joita sääntöjen on tarkoitus kutsua. Sääntöluokissa määriteltävät funktiot toimivat samoin kuin Java-luokissa. Ne ovat toimintoja, jotka palauttavat jonkin arvon niille syötetyn parametrin perusteella. Kyselyt (engl. *Queries)* ovat funktioita, joiden avulla voidaan hakea sääntömoottorin työmuistissa olevia muuttujia.

Varsinaisten liiketoimintasääntöjen rakenne .drl-tiedostoissa noudattaa seuraavaa kaavaa:

1. Säännön nimi.

2. Säännön attribuutit.

3. Liiketoimintasäännön ehtolauseke, joka sisältää arvioitavan ehdon. Ehtolauseke aloitetaan .drl-tiedostoissa *when*-avainsanalla.

4. Liiketoimintasäännön seurauslauseke, joka kertoo mitä tapahtuu kun *when*-avainsanalla aloitetussa ehtolausekkeessa määritelty ehto toteutuu. Seurauslauseke aloitetaan *then*-avainsanalla.

5. Liiketoimintasäännön päättävä *end*-avainsana.

Liiketoimintasäännöille voidaan antaa lukuisia attribuutteja, jotka vaikuttavat niiden toimintaan. Joitakin seuraavista attribuutti-esimerkeistä hyödynnetään myös opinnäytetyön toteutusvaiheessa:

- DATE-EFFECTIVE. Säännöt voivat olla aikaan sidottuja, jolloin niiden voimassaolo liittyy johonkin päivämäärään tai ne ovat voimassa tiettynä ajanjaksona. Date-effective-attribuutti määrittelee päivämäärän, josta alkaen sääntö on voimassa.

- DATE-EXPIRES. Tämä attribuutti liittyy myös säännön voimassaolon määrittelyyn, tarkemmin säännön vanhenemispäivämäärään. Sääntö

aktivoituu vain, kun nykyinen päivämäärä on ennen attribuutissa määriteltyä päivämäärää. Yhdistämällä date-effective- ja date-expires-attribuutti saadaan siis määriteltyä ajanjakso, jolloin säännöt ovat voimassa.

- NO-LOOP. Kun ohjelman ajon aikana muutetaan sääntömoottorin työmuistissa olevaa tietoa kutsumalla modify- tai update-metodeja, säännöt aktivoituvat ja sääntömoottori vertaa niitä uudelleen muutettuun tietoon (faktoihin). Tämän seurauksena voi syntyä loputon kierre, jossa jonkin säännön ehto täyttyy ja sessiossa olevaa tietoa päivitetään yhä uudelleen. No-loop-attribuutti estää säännön aktivoitumisen uudelleen.

- RULEFLOW-GROUP. Jos sääntö kuuluu johonkin sääntöjoukkoon, niille voidaan määritellä yhteinen nimi. Sääntöjoukkoon kuuluvat säännöt ajetaan ainoastaan silloin kun niiden sääntöjoukkoa kutsutaan. Tätä attribuuttia käytetään erityisesti prosessien suunnittelussa, jolloin tietyssä prosessin vaiheessa aktivoidaan jokin tietty sääntöjoukko.

- SALIENCE. Säännön keskeisyys ilmaisee säännön tärkeyden ja sen avulla voidaan määrittää, missä järjestyksessä säännöt aktivoidaan. Salience()-metodilla on yksi parametri, jonka on oltava joko positiivinen tai negatiivinen kokonaisluku. Mitä suurempi arvo on, sitä todennäköisemmin sääntö aktivoituu ensin. Oletusarvo säännön keskeisyydelle on 0. Keskeisyydelle voidaan antaa myös dynaaminen arvo, jolloin keskeisyys riippuu jostakin muuttujasta. Esimerkkinä on tapaus, jossa säännön tärkeys riippuu tilin saldosta:

```
salience((int)$tili.getSaldo())
```

Sääntömoottorin työmuistissa oleviin olioihin voidaan viitata muuttujina sääntötiedostojen sisältä. Muuttujiin voi tallentaa esimerkiksi laskutoimituksien tai sääntöjen välituloksia, ja liiketoimintasäännöt voivatkin toimia niin, että prosessin edellisessä vaiheessa tallennettuun muuttujaan viitataan prosessin

seuraavassa vaiheessa. Muuttujiin viittaaminen Droolsissa tapahtuu usein $-merkin avulla, joka ei ole pakollinen, mutta helpottaa muuttujien erottumista.

Sääntöjen seuraukset, jotka määritellään *then*-avainsanan jälkeen, voivat olla esimerkiksi kutsuja metodeihin tai funktioihin. Seurauslausekkeessa voidaan myös vaikuttaa sääntömoottorin työmuistissa oleviin olioihin. Työmuistissa olevia tietoja voidaan hallita seuraavilla metodeilla:

- **Modify/Update**: Sääntömoottorin työmuistissa olevaa tietoa voidaan muuntaa ja päivittää.

- **Insert**: Työmuistiin voidaan lisätä uusia tietoja, joita saatetaan tarvita myöhemmässä vaiheessa.

- **Retract**: Työmuistista voidaan poistaa tietoa, jos esimerkiksi ei haluta minkään muun säännön aktivoituvan kyseisen tiedon perusteella.

Droolsin sääntömoottori pystyy lukemaan liiketoimintasääntöjä myös suoraan taulukkotiedostoista (kuva 6). Taulukkojen rakenne on tarkkaan määritelty, ja väärin formatoitu taulukko aiheuttaa ongelmia säännöstön lukemisessa. Taulukkomuotoisissa sääntötiedostoissa yksi rivi edustaa yhtä liiketoimintasääntöä, eli taulukot sopivat hyvin tilanteisiin, joissa säännöillä on samanlainen rakenne, mutta säännöissä käytettävät parametrit vaihtuvat. Taulukkomuotoisissa sääntötiedostoissa käytetään pääosin samoja avainsanoja kuin DRL-tiedostoissa, mutta esimerkiksi ehtolauseen ja seurauslausekkeen avainsanat vaihtuvat muotoon CONDITION ja ACTION, eli ehto ja toiminto.

	RuleSet	com.sample			
	Import	data.Tuotetyyppi			
	Import	data.Alvluokitus			
	Import	data.Tuote			
	NO-LOOP	true			
	RULEFLOW-GROUP	alvluokitus			
	Notes	Sääntötaulukko ALV-luokituksen vaihtamista varten			
	RuleTable ALV-luokitukset				
	CONDITION		RULEFLOW-GROUP	NO-LOOP	ACTION
	$tuote:Tuote				
	tuotetyyppi.equals($param)				$tuote.setAlvluokitus($param); update($tuote)
	ALV-säännöt	Tyyppi			Aseta ALV-luokitus
	Yleinen verokanta	Tuotetyyppi.YLEINEN	alvluokitus	true	Alvluokitus.YLEINEN
	Alennettu 1: Elintarvike	Tuotetyyppi.ELINTARVIKE	alvluokitus	true	Alvluokitus.ALENNETTU1
	Alennettu 1: Rehut	Tuotetyyppi.REHUT	alvluokitus	true	Alvluokitus.ALENNETTU1
	Alennettu 1: Ravintola - ja ateriapalvelut	Tuotetyyppi.RAVINTOLAPALVELU	alvluokitus	true	Alvluokitus.ALENNETTU1
	Alennettu 2: Kirjat	Tuotetyyppi.KIRJA	alvluokitus	true	Alvluokitus.ALENNETTU2
	Alennettu 2: Sanomalehdet	Tuotetyyppi.LEHTI	alvluokitus	true	Alvluokitus.ALENNETTU2
	Alennettu 2: Lääkkeet	Tuotetyyppi.LAAKE	alvluokitus	true	Alvluokitus.ALENNETTU2
	Alennettu 2: Liikuntapalvelut	Tuotetyyppi.LIIKUNTAPALVELU	alvluokitus	true	Alvluokitus.ALENNETTU2
	Alennettu 2: Elokuvanäytökset	Tuotetyyppi.ELOKUVANAYTOS	alvluokitus	true	Alvluokitus.ALENNETTU2
	Alennettu 2: Kulttuuritilaisuudet	Tuotetyyppi.KULTTUURITILAISUUS	alvluokitus	true	Alvluokitus.ALENNETTU2
	Alennettu 2: Henkilökuljetukset	Tuotetyyppi.HENKILOKULJETUS	alvluokitus	true	Alvluokitus.ALENNETTU2
	Alennettu 2: Majoituspalvelut	Tuotetyyppi.MAJOITUSPALVELU	alvluokitus	true	Alvluokitus.ALENNETTU2
	Alennettu 2: Televisio- ja yleisradiotoiminnan korvaukset	Tuotetyyppi.TELEVISIO_RADIO_KORVAUS	alvluokitus	true	Alvluokitus.ALENNETTU2
	Nollaverokanta: Tavaran myynti EU ALV-velvollisille	Tuotetyyppi.MYYNTI_EU_ALV	alvluokitus	true	Alvluokitus.NOLLAVEROKANTA
	Nollaverokanta: Vienti EU:n ulkopuolelle	Tuotetyyppi.VIENTI_EU_ULK	alvluokitus	true	Alvluokitus.NOLLAVEROKANTA
	Nollaverokanta: Jäsenlehdet yhteisöille	Tuotetyyppi.MAINOS_LEHDET_YHTEISO	alvluokitus	true	Alvluokitus.NOLLAVEROKANTA

Kuva 6. Säännöstö Excel-tiedostona.

Taulukkotiedoston käyttäminen säännöstön ylläpitoon on Droolsin matalan tason low-code-ratkaisu. Drools tarjoaa myös muita graafisia käyttöliittymiä sääntöjen muokkaamiseen kuten Drools Workbenchin. Taulukkotiedostoja pystyy kuitenkin käsittelemään esimerkiksi Eclipse-kehitysympäristön sisällä sisäänrakennetun Excel-editorin avulla, jonka vuoksi niitä käytetään opinnäytetyön toteutusvaiheessa.

4 Toteutettavan prototyypin kuvaus

Toteutusprojektissa sovelletaan sääntömoottorin käyttöä tilanteessa, jossa pyritään selvittämään Kelan asiakkaan oikeus eläkkeensaajan asumistukeen sekä myönnettävän tuen määrä. Asumistuen määrä on 85 prosenttia huomioon otettavista asumismenoista, joista on vähennetty asumismenojen omavastuuosuus, eli perusomavastuu ja tulojen mukaan määräytyvä lisäomavastuu. Eläkkeensaajan asumistuki lasketaan siis seuraavan kaavan mukaan:

$$0{,}85 \text{ x } (\text{huomioon otettavat asumismenot} - (\text{perusomavastuu} + \text{mahdollinen lisäomavastuu}))$$

Tässä luvussa määritellään laskennassa käytettävät tiedot ja rajataan ne prototyypille sopivaksi. Tiedot on koottu eläkkeensaajan asumistuen määräytymisperusteiden pohjalta, jotka määritellään laissa eläkkeensaajan asumistuesta luvussa 2 (Laki eläkkeensaajan asumistuesta 2007/571) sekä lakia tarkentavassa valtioneuvoston asetuksessa (Valtioneuvoston asetus eläkkeensaajan asumistuen määräytymisperusteista vuonna 2023). Laskentaa varten on myös etsitty tietoa Kelan verkkosivuilta (Eläkkeensaajan asumistuen määrän laskentaohje ja Eläkkeensaajan asumistuki). Projektissa käytetään vuodelle 2023 säädettyjä määräytymisperusteita.

4.1 Oikeus asumistukeen

Ennen varsinaisen laskennan suorittamista on ensin selvitettävä asiakkaan oikeus asumistukeen. Lain mukaan oikeus eläkkeensaajan asumistukeen on hakijalla, joka on täyttänyt 16 vuotta ja joka saa jotakin etuuteen oikeuttavista eläkkeistä. Eläkkeensaajan asumistukeen oikeuttavat eläkkeet luetellaan projektissa yksinkertaistetussa muodossa, sillä laskentaprosessissa aikana tapahtuvan tarkistuksen kannalta olennaista on ainoastaan eläkkeen tai etuuden tyyppi. Eläkkeensaajan asumistukeen oikeuttavia etuuksia ovat vanhuuseläke, työkyvyttömyyseläke, leskeneläke, takuueläke, työuraeläke, jatkuva tapaturmaeläke, elinkorko, huoltoeläke, ansionmenetyksen korvaus, jatkuva kuntoutusraha, jatkuva työkyvyttömyyseläke tai jokin lain momentissa mainittua etuutta vastaava ulkomailta maksettava etuus. (Laki eläkkeensaajan asumistuesta 2007/571.)

4.2 Laskennassa käytettävät lähtötiedot

Laskennan suorittamiseksi hakijalta ja tämän mahdolliselta puolisolta vaaditaan tietoja. Tietojen perusteella luodaan tietorakenteita, eli tämän projektin tapauksessa Java-luokkia, jotka edustavat hakijaa ja tämän puolisoa tietojärjestelmässä. Eläkkeensaajan asumistuen laskemiseksi sekä hakijalta että tämän puolisolta vaadittavia tietoja ovat ikä, mahdolliset saatavat etuudet ja tulojen, omaisuuden ja velan määrä. Lisäksi vaaditaan myös hakijan ja puolison

yhteisen asunnon tiedot, joita ovat sijaintikunta, asuntotyyppi ja asunnosta aiheutuvat asumismenot. Jos asunto on omakotitalo, vaaditaan hoitomenojen laskemista varten tiedot myös asunnon pinta-alasta ja valmistumis- tai perusparannusvuodesta.

4.3 Laskennan parametrit

Laskentaprosessissa käytettävät tiedot voidaan jakaa kolmeen kategoriaan: osa tiedoista tulee suoraan hakijalta, osa tiedoista on laissa määriteltyjä vakioita ja osa taas määräytyy hakijan antamien tietojen perusteella.

Vakioilla tarkoitetaan tässä yhteydessä huomioon otettavia parametreja, jotka ovat kaikille hakijoille samat, riippumatta lähtötiedoista. Laissa määriteltyjä vakioita ovat:

- Eläkkeensaajan asumistuen määrä on 85 prosenttia huomioon otettavista asumismenoista, joista on vähennetty perusomavastuu ja tulojen mukaan määräytyvä lisäomavastuu. Tämä muuttuu prosenttikertoimeksi 0,85.

- Perusomavastuu on kaikille hakijoille sama 681,39 euroa / vuosi eli noin 56,78 euroa / kuukausi.

- Lisäomavastuu on 41,3 prosenttia siitä perheen tulojen osasta, joka ylittää tulorajan. Tämä muuttuu prosenttikertoimeksi 0,413.

- Tuloihin lisätään 8 prosenttia siitä omaisuuden osasta, joka ylittää omaisuusrajan. Tämä muuttuu prosenttikertoimeksi 0,08.

- Omakotitalon hoitomenoina otetaan huomioon vesimaksu, jonka suuruus on henkilöä kohden 32,40 e/kk.

Vakioiden lisäksi laskennassa käytetään parametreja, jotka perustuvat hakijan ja tämän puolison luovuttamiin tietoihin. Parametrien välillä on myös keskinäisiä riippuvuuksia.

- Kunnossapitokustannuksien enimmäismäärään vaikuttaa asunnon valmistumis- tai perusparannusvuosi.
    - Jos asunto on valmistunut tai peruskorjattu vuonna 1974 tai sen jälkeen, kunnossapitokustannukset ovat 47,84 e/kk.
    - Jos asunto on valmistunut tai peruskorjattu ennen vuotta 1974, kunnossapitokustannukset ovat 62,19 e/kk.
- Omakotitalon lämmityskuluja korvataan vain asunnon kohtuullisen koon mukaan.
    - Yksin asuvalle kohtuullinen asunnon koko on enintään 70 m².
    - Pariskunnalle asunnon kohtuullinen koko on enintään 85 m².
- Asunnon lämmityskustannuksiin vaikuttavat asunnon lämmitysryhmä sekä asunnon valmistumis- tai perusparannusvuosi. Asunnon lämmitysryhmä puolestaan määräytyy sijaintikunnan mukaan. Jos asunto on valmistunut vuonna 1974 tai sen jälkeen, lämmitysmenot ovat seuraavat:
    - Lämmitysryhmä 1: 2,07 e / m² / kk
    - Lämmitysryhmä 2: 2,20 e / m² / kk
    - Lämmitysryhmä 3: 2,35 e / m² / kk

  Jos taas asunto on valmistunut ennen vuotta 1974, käytetään seuraavia lämmitysmenoja:

  - Lämmitysryhmä 1: 2,70 e / m² / kk
  - Lämmitysryhmä 2: 2,86 e / m² / kk
  - Lämmitysryhmä 3: 3,06 e / m² / kk

- Asumismenoilla on enimmäismäärä, joka määräytyy asunnon kuntaryhmän mukaan. Kuntaryhmä riippuu asunnon sijaintikunnasta. Kuntaryhmiä on kolme:
  - Kuntaryhmän 1 asumismenojen enimmäismäärä on 9 287 e/v eli noin 774 e/kk.
  - Kuntaryhmän 2 asumismenojen enimmäismäärä on 8 541 e/v eli noin 712 e/kk.
  - Kuntaryhmän 3 asumismenojen enimmäismäärä on 7 493 e/v eli noin 624 e/kk.

- Lisäomavastuu on 41,3 prosenttia siitä perheen tulojen osasta, joka ylittää tulorajan. Jos tulot jäävät alle tulorajan, lisäomavastuuta ei tule. Lisäomavastuun tulorajat perhesuhteiden mukaan vuonna 2023 ovat seuraavat:
  - Yksin asuvalle 10 280 euroa / vuosi eli noin 856,67 euroa / kuukausi.
  - Avio- tai avoliitossa tai rekisteröidyssä parisuhteessa olevalle, jonka puolisolla ei ole oikeutta asumistukeen 14 746 euroa / vuosi eli noin 1228,80 euroa / kuukausi.
  - Avio- tai avoliitossa tai rekisteröidyssä parisuhteessa olevalle, jonka puolisolla on oikeus asumistukeen 16 783 euroa / vuosi eli noin 1398,58 euroa / kuukausi.

- Tuloihin lisätään 8 % siitä omaisuuden osasta, joka ylittää omaisuusrajan. Omaisuusrajat perhesuhteiden mukaan ovat:
    - Yksinäiselle henkilölle 18 306 euroa.
    - Avio- tai avopuolisoille yhteensä 29 290 euroa.
- Pienin maksettava asumistuki on 7,46 euroa / kk. Jos molemmat puolisot saavat asumistukea, pienin maksettava asumistuki on 3,73 euroa /kk.

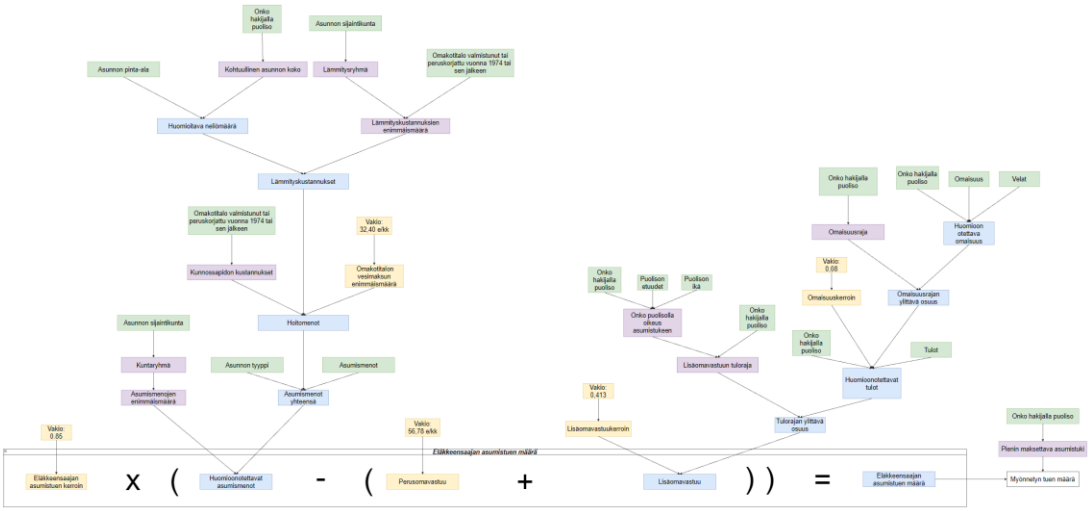
5 Liiketoimintalogiikan toteutus

Toteutuksen aloituksessa kiinnitetään huomiota kahteen sovellukselta vaadittavaan asiaan:

1. Sovelluksen täytyy selvittää asiakkaan (ja tämän puolison) oikeus asumistukeen.
2. Sovelluksen täytyy laskea eläkkeensaajan asumistuen määrä.

Vaatimukset yhdistetään yhdeksi prosessiksi, jonka tuloksena on päätös asumistuen myöntämisestä sekä mahdollisen myönnetyn etuuden määrä. Koska hakijan oikeus asumistukeen on helposti selvitettävissä hakijalta saatujen tietojen perusteella, keskitytään aluksi eläkkeensaajan asumistuen laskentakaavaan.

Laskentakaava on rakenteeltaan yksinkertainen, mutta laskennassa käytettävien muuttujien taakse kätkeytyy lukuisia muita muuttujia, laskutoimituksia ja liiketoimintasääntöjen avulla tehtäviä päätöksiä. Kuvassa 7 on esitetty kaikkien laskentakaavaan sisältyvien tietojen keskinäiset riippuvuudet.

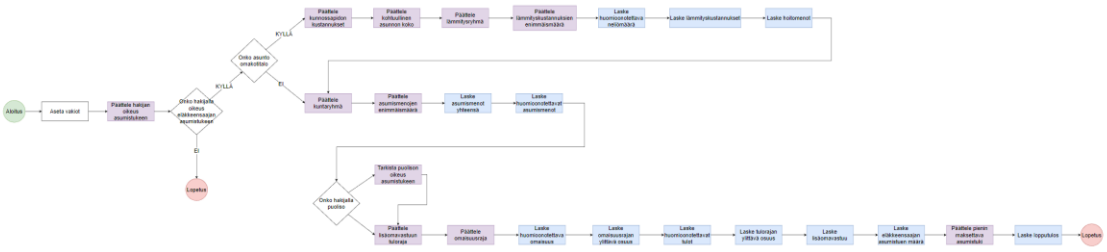


Kuva 7. Eläkkeensaajan asumistuen muuttujien keskinäiset riippuvuudet.

UML-notaatiolla tehty mallinnus kuvaa eri tietojen välisiä suhteita sekä niiden alkuperää. Keltaisella merkityt muuttujat ovat luvussa 4 lueteltuja vakioita, jotka ovat kaikille hakijoille samat. Vakiot voidaan siis asettaa oikeisiin arvoihinsa heti laskentaprosessin aluksi. Vihreällä merkityt muuttujat puolestaan tarkoittavat tietoja, jotka tulevat hakijalta itseltään. Sovelluksen liiketoimintasääntöjen toiminta perustuu näihin kahteen muuttujatyyppiin. Liiketoimintasääntöjen läpikäynnin tuloksena syntyvät tiedot on merkitty violetilla. Ennen varsinaisen eläkkeensaajan asumistuen määrän laskemista joudutaan myös laskemaan muita laskutoimituksia. Sinisellä merkittyjen muuttujien arvo on tulos jostakin laskutoimituksesta.

5.1 Laskentaprosessin mallinnus

Kun laskennassa käytettävien muuttujien väliset suhteet ovat selvillä, voidaan ryhtyä mallintamaan prosessia. Prosessin muodostuksessa kiinnitetään edelleen huomiota muuttujien keskinäisiin riippuvuuksiin: prosessin tietyssä vaiheessa vaadittavan muuttujan arvo täytyy selvittää ennen kyseistä operaatiota. Kuvassa 8 on esitetty laskentaprosessin malli.

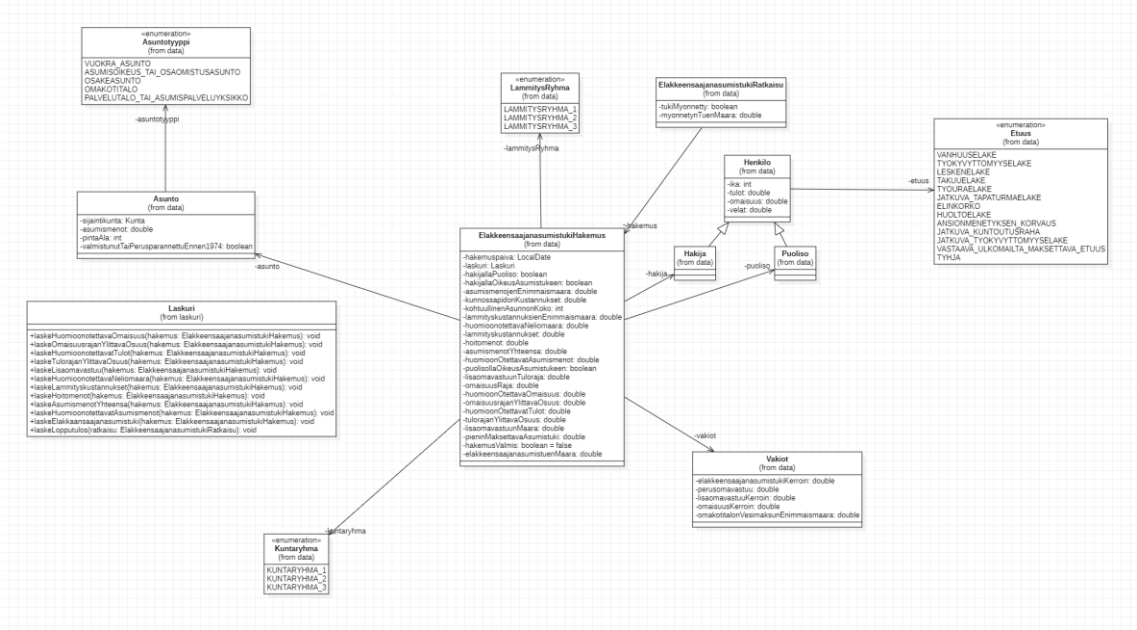


Kuva 8. Laskentaprosessin UML-mallinnus.

Kuvassa puhutaan muuttujien sijasta päätelmistä ja laskutoimituksista eli operaatioista. Värikoodit viittaavat kuitenkin samoihin asioihin: Violetilla pohjalla ovat operaatiot, jotka suoritetaan liiketoimintasääntöjen avulla ja sinisellä pohjalla ovat laskutoimitukset. Värittömät ruudut ovat prosessin hallintaan liittyviä portteja, jotka määrittävät prosessin etenemissuunnan jonkin ehdon perusteella.

5.2 Laskennassa käytettävät tietorakenteet

Luvussa 4 kuvatun prototyypin pohjalta voidaan muodostaa laskennassa käytettävät tietorakenteet. Tietorakenteet ovat tyypillisiä Java-luokkia ja arvojoukkoja. Kuvassa 9 on esitetty laskennassa käytettävien tietorakenteiden luokkakaavio, josta ilmenee ainoastaan POJO-luokkien väliset suhteet.



Kuva 9. Laskennassa käytettävien tietorakenteiden luokkakaavio.

ElakkeensaajanasumistukiRatkaisu-luokka toimii kaikki tiedot kokoavana pääluokkana. Toteutuksien ydinajatuksena onkin, että prosessiin syötetään ainoastaan ratkaisuolio, joka sisältää tiedot ratkaisun tilasta ja myönnetyn etuuden määrästä, sekä sen luokkamuuttujaksi määritelty ElakkeensaajanasumistukiHakemus-olio, joka muodostaa ratkaisulle perusteet. Kuvassa 9 poikkeuksen muodostaa Laskuri-luokka, jota ei käytetä tiedon varastointiin, vaan pelkästään laskutoimituksien suorittamiseen (esimerkkikoodi 2). Laskuri-luokkaa käytetään molemmissa toteutuksissa ja se liitetään suoraan EläkkeensaajanasumistukiHakemus-olioon luokkamuuttujana. Laskuri-luokka sisältää yhteensä 12 metodia, jotka toteuttavat prosessin kaikki laskutoimitukset niille annettujen parametrien perusteella. Jotta eri toteutustapojen erot voidaan selkeästi osoittaa, molemmissa toteutuksissa käytetään samoja tietoluokkia ilman toteutustapakohtaisia muutoksia tai täydennyksiä.

```
102    public void laskeElakkaansaajanasumistuki(ElakkeensaajanasumistukiHakemus hakemus) {
103        Vakiot vakiot = hakemus.getVakiot();
104        double maara = vakiot.getElakkeensaajanasumistukiKerroin() * (hakemus.getHuomioonOtettavatAsumismenot()
105                - (vakiot.getPerusomavastuu() + hakemus.getLisaomavastuunMaara()));
106
107        hakemus.setElakkeensaajanasumistuenMaara(maara);
108    }
```

Esimerkkikoodi 2. Eläkkeensaajan asumistuen määrän laskeva metodi.

5.3 Testitapaukset

Kun laskentaprosessi on saatu muodostettua, luodaan testitapaukset. Testitapaukset ovat kuviteltuja etuuden hakijoita, joilla on erilaisia ominaisuuksia. Jotta testitapaukset olisivat kattavia, niitä määritellään useampi, ja niiden ominaisuuksien tulee olla mahdollisimman monipuolisia. Testitapaukset eivät ole siis yksikkötestejä, vaan ne tarjoavat tavan testata laskentaprosessia isommassa mittakaavassa.

Testitapaus 1. Hakija on 34-vuotias henkilö, joka ei saa eläkkeensaajan asumistukeen oikeuttavaa eläkettä tai etuutta. Testitapauksella ei lain mukaan ole oikeutta eläkkeensaajan asumistukeen. Kts. tarkemmat tiedot liitteestä 1.

Testitapaus 2. Hakija on 65-vuotias henkilö, joka saa eläkkeensaajan asumistukeen oikeuttavaa vanhuuseläkettä. Testitapauksella on lain mukainen oikeus eläkkeensaajan asumistukeen. Kts. tarkemmat tiedot liitteestä 2.

Testitapaus 3. Hakija on 52-vuotias henkilö, joka saa eläkkeensaajan asumistukeen oikeuttavaa takuueläkettä. Testitapauksella on myös oikeus eläkkeensaajan asumistukeen. Kts. tarkemmat tiedot liitteestä 3.

Testitapaus 4. Hakija 45-vuotias henkilö, joka saa eläkkeensaajan asumistukeen oikeuttavaa työkyvyttömyyseläkettä. Testitapauksella on oikeus eläkkeensaajan asumistukeen. Hakijalla ei ole puolisoa. Kts. tarkemmat tiedot liitteestä 4.

5.4 Toteutus ilman sääntömoottoria

Ilman sääntömoottoria tapahtuvassa ohjelmointitoteutuksessa ohjelman rakenne suunnitellaan mahdollisimman yksinkertaiseksi. Tavoitteena on, että aiemmin määritellyn prosessin kaikki vaiheet sekä sovelluksen liiketoimintasäännöt ovat selkeästi nähtävillä.

Sovelluksella on yksi pääluokka, Laskentasovellus, jonka tarkoituksena on toimia sovelluksen runkona (esimerkkikoodi 3). Laskentasovellus-luokalla on ainoastaan yksi metodi, jota kutsumalla koko prosessi käydään läpi. Prosessin eri vaiheet on lisätty apuluokkien kutsuina kyseisen metodin alle, eli sen tarkoitus on simuloida prosessia. Metodille annetaan parametriksi ElakkeensaajanasumistukiRatkaisu-tyyppinen olio, jonka sisällä olevaan ElakkeensaajanasumistukiHakemus-olioon tallennetaan prosessin kaikki välitulokset.

Laskentasovellus.java

```java
package laskentasovellus;

import data.Asuntotyyppi;
import data.ElakkeensaajanasumistukiHakemus;
import data.ElakkeensaajanasumistukiRatkaisu;
import laskuri.Laskuri;

public class Laskentasovellus {

    ElakkeensaajanasumistukiHakemus hakemus;
    ElakkeensaajanasumistukiRatkaisu ratkaisu;
    PaattelyMoottori paattelyMoottori;
    Laskuri laskuri;



    // Laskentasovellus simuloi prosessia
    public void teeRatkaisu(ElakkeensaajanasumistukiRatkaisu ratkaisu) {

        this.ratkaisu = ratkaisu;
        this.hakemus = this.ratkaisu.getHakemus();
        this.paattelyMoottori = new PaattelyMoottori();
        this.laskuri = new Laskuri();


        // 1. Asetetaan hakemuksessa käytettävät vakiot
        paattelyMoottori.paatteleVakiot(hakemus);


        // 2. Tarkistetaan onko hakijalla oikeus asumistukeen
        paattelyMoottori.paatteleOnkoHakijallaOikeusAsumistukeen(hakemus);


        // Prosessin ensimmäinen haara
        if(hakemus.isHakijallaOikeusAsumistukeen()) {


            // Prosessin toinen haara
            if(hakemus.getAsunto().getAsuntotyyppi().equals(Asuntotyyppi.OMAKOTITALO)) {
                paattelyMoottori.paatteleKunnossapidonKustannukset(hakemus);
                paattelyMoottori.paatteleKohtuullinenAsunnonKoko(hakemus);
                paattelyMoottori.paatteleLammitysryhma(hakemus);
                paattelyMoottori.paatteleLammityskustannuksienEnimmaismaara(hakemus);
                laskuri.laskeHuomioonotettavaNeliomaara(hakemus);
                laskuri.laskeLammityskustannukset(hakemus);
                laskuri.laskeHoitomenot(hakemus);
            }

            paattelyMoottori.paatteleKuntaryhma(hakemus);
            paattelyMoottori.paatteleAsumismenojenEnimmaismaara(hakemus);
            laskuri.laskeAsumismenotYhteensa(hakemus);
            laskuri.laskeHuomioonotettavatAsumismenot(hakemus);

            // Prosessin kolmas haara
            if(hakemus.isHakijallaPuoliso()) {
                paattelyMoottori.paatteleOnkoPuolisollaOikeusAsumistukeen(hakemus);
            }

            paattelyMoottori.paatteleLisaomavastuunTuloraja(hakemus);
            paattelyMoottori.paatteleOmaisuusRaja(hakemus);
            laskuri.laskeHuomioonotettavaOmaisuus(hakemus);
            laskuri.laskeOmaisuusrajanYlittavaOsuus(hakemus);
            laskuri.laskeHuomioonotettavatTulot(hakemus);
            laskuri.laskeTulorajanYlittavaOsuus(hakemus);
            laskuri.laskeLisaomavastuu(hakemus);
            laskuri.laskeElakkaansaajanasumistuki(hakemus);
            paattelyMoottori.paattelePieninMaksettavaAsumistuki(hakemus);
            laskuri.laskeLopputulos(ratkaisu);

        }

    }
}
```

Esimerkkikoodi 3. Laskentasovelluksen runko.

Laskentasovellus kutsuu prosessin aikana kahta apuluokkaa, laskuria ja päättelymoottoria. Tässä yhteydessä päättelymoottori ei tarkoita oikeaa sääntömoottorin sisällä päätöksiä tekevää rakennetta, vaan Paattelymoottori-luokan tarkoituksena on imitoida sääntömoottorin toimintaa implementoimalla liiketoimintasäännöt. Ohjelmointitoteutuksessa se sisältää vastaavasti liiketoimintasääntöjä hyödyntäviä metodeja (esimerkkikoodi 4).

```
234    public void paatteleLammityskustannuksienEnimmaismaara(ElakkeensaajanasumistukiHakemus hakemus) {
235
236        if (Util.sisaltyyAjanjaksoon(hakemus.getHakemuspaiva(), alkupvm, loppupvm)) {
237
238            if (hakemus.getAsunto().isValmistunutTaiPerusparannettuEnnen1974()) {
239
240                if (hakemus.getLammitysRyhma().equals(LammitysRyhma.LAMMITYSRYHMA_1)) {
241                    hakemus.setLammityskustannuksienEnimmaismaara(2.70);
242                } else if (hakemus.getLammitysRyhma().equals(LammitysRyhma.LAMMITYSRYHMA_2)) {
243                    hakemus.setLammityskustannuksienEnimmaismaara(2.86);
244                } else if (hakemus.getLammitysRyhma().equals(LammitysRyhma.LAMMITYSRYHMA_3)) {
245                    hakemus.setLammityskustannuksienEnimmaismaara(3.06);
246                }
247
248            } else {
249
250                if (hakemus.getLammitysRyhma().equals(LammitysRyhma.LAMMITYSRYHMA_1)) {
251                    hakemus.setLammityskustannuksienEnimmaismaara(2.07);
252                } else if (hakemus.getLammitysRyhma().equals(LammitysRyhma.LAMMITYSRYHMA_2)) {
253                    hakemus.setLammityskustannuksienEnimmaismaara(2.20);
254                } else if (hakemus.getLammitysRyhma().equals(LammitysRyhma.LAMMITYSRYHMA_3)) {
255                    hakemus.setLammityskustannuksienEnimmaismaara(2.35);
256                }
257
258            }
259
260        }
261    }
262
```

Esimerkkikoodi 4. Liiketoimintasäännöt Java-kielellä.

Esimerkkikoodissa 3 esitetyssä metodissa päätellään lämmityskustannuksien enimmäismäärä. Liiketoimintasäännöt ovat usein aikaan sidottuja, ja tämä on otettu huomioon myös ohjelmointitoteutuksessa. Liiketoimintasäännön ensimmäinen ehtolause tarkistaa, että hakemuspäivämäärä sisältyy käynnissä olevaan ajanjaksoon. Seuraavaksi tarkistetaan, onko asunto rakennettu tai perusparannettu ennen vuotta 1974, ja sen jälkeen lämmityskustannuksien enimmäismenot asetetaan lämmitysryhmän mukaan. Lämmitysryhmä on puolestaan johdettu asunnon sijaintikunnasta prosessin aiemmassa vaiheessa. Esimerkkikoodista 3 on havaittavissa liiketoimintasäännön toiminta ohjelmakoodin seassa. Luvussa 2 esitetty liiketoimintasääntöjen rakenne toistuu esimerkissä useasti: IF <ehtolauseke> THEN <seurauslauseke> ELSE

<seurauslauseke>. Näiden ehtolausekkeiden avulla muuttujille saadaan päätteltyä niiden oikeat arvot.

Laskentasovellus ajetaan testitapausten avulla, jotka on koottu erilliseen testitiedostoon (esimerkkikoodi 5). Testit on toteutettu JUnit-testikirjaston avulla. Sovellus alustetaan uudelleen ennen jokaista testitapausta ja itse testeissä sovellukselle syötetään vaadittavat lähtötiedot ja käynnistetään laskentaprosessi teeRatkaisu()-metodikutsun avulla.

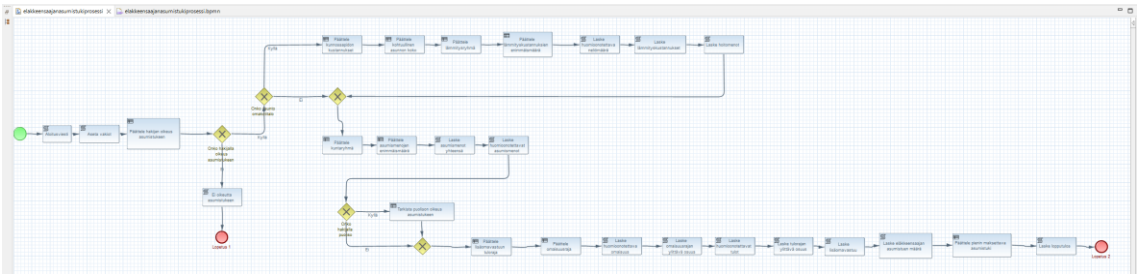
LaskentasovellusTest.java

```
56
57      @Test
58      void testitapaus2() {
59
60          // Hakijan tiedot
61          hakija.setIka(67);
62          hakija.setEtuus(Etuus.VANHUUSELAKE);
63          hakija.setTulot(600);
64          hakija.setOmaisuus(1400);
65          hakija.setVelat(0);
66
67          // Liitetään hakija hakemukseen
68          hakemus.setHakija(hakija);
69
70          // Puolison tiedot
71          puoliso.setIka(65);
72          puoliso.setEtuus(Etuus.VANHUUSELAKE);
73          puoliso.setTulot(500);
74          puoliso.setOmaisuus(3200);
75          puoliso.setVelat(0);
76
77          // Liitetään puolison tiedot hakemukseen
78          hakemus.setPuoliso(puoliso);
79          hakemus.setHakijallaPuoliso(true);
80
81          // Asunnon tiedot
82          Asunto asunto = new Asunto();
83          asunto.setSijaintikunta(Kunta.HELSINKI);
84          asunto.setAsuntotyyppi(Asuntotyyppi.VUOKRA_ASUNTO);
85          asunto.setAsumismenot(770);
86
87          // Liitetään asunnon tiedot hakemukseen
88          hakemus.setAsunto(asunto);
89
90          // Liitetään hakemus ratkaisupohjaan
91          ratkaisu.setHakemus(hakemus);
92
93          // Prosessoidaan hakemus
94          Laskentasovellus laskentasovellus = new Laskentasovellus();
95          laskentasovellus.teeRatkaisu(ratkaisu);
96
97          // Tulokset
98          assertEquals(303.11, ratkaisu.getMyonnetynTuenMaara());
99          assertEquals(true, ratkaisu.isTukiMyonnetty());
100     }
```

Esimerkkikoodi 5. Ohjelmointitoteutuksen testaus.

5.5 Toteutus Drools-sääntömoottorin avulla

Kun sovellusta ryhdytään toteuttamaan sääntömoottorin avulla, voidaan lähestymistavan huomata olevan jo lähtökohtaisesti lähempänä prosesseja ja liiketoimintasääntöjä. Toteutus on kaksi keskeistä osaa: liiketoimintasääntöjen, eli tässä tapauksessa .xls-muotoisten taulukkojen sekä liiketoimintaprosessin muodostaminen. Laskentaprosessi mallinnetaan pala palalta uudelleen käyttämällä joko Droolsin mukana tulevaa BPMN-työkalua tai kehittyneempää BPMN2 Modeler -lisäosaa (kuva 10). Tällä kertaa kyseessä ei ole kuitenkaan pelkkä mallinnus, vaan pikemminkin sovelluksen visuaalinen ilmentymä, joka kuvaa tarkalleen sen, mitä komponentteja sovelluksesta löytyy. Sääntömoottori käyttää mallinnettua prosessia ohjelman suorittamiseen.

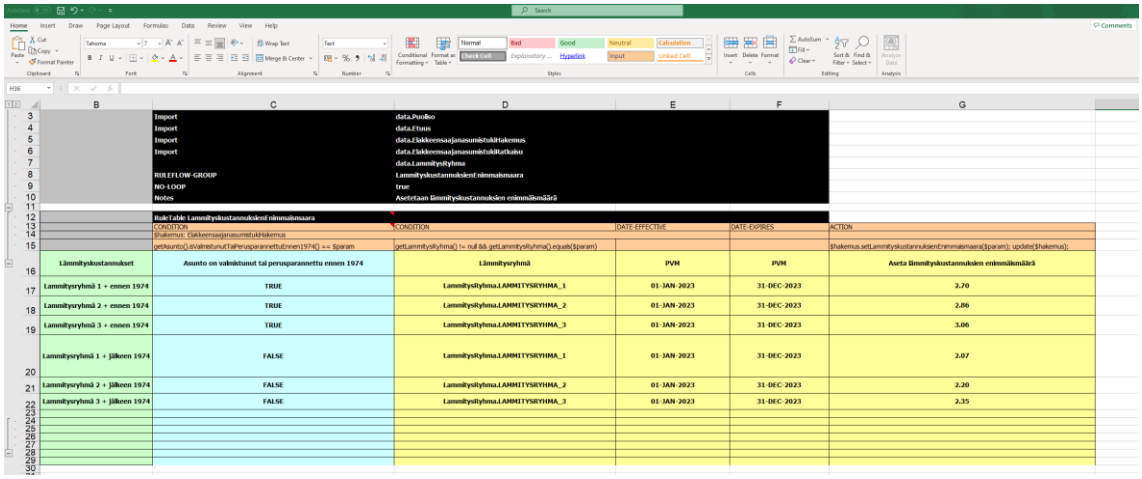


Kuva 10. Laskentaprosessin BPMN2-mallinnus.

Prosessin operaatiot on määritelty joko *Rule Taskeiksi* eli sääntötiedostoja hyödyntäviksi operaatioiksi tai *Script Taskeiksi* eli ohjelmakoodia hyödyntäviksi operaatioiksi. Sääntö-operaatioihin siirryttäessä kutsutaan tiettyyn sääntöjoukkoon kuuluvia sääntöjä, ja ohjelmakoodi-operaatioissa puolestaan suoritetaan jokin määritelty ohjelmakoodi. Mallinnuksessa näkyy myös prosessin etenemissuuntaa ohjaavia portteja, jotka ovat keltaisella pohjalla. Loogiset portit ohjaavat prosessin tiettyyn suuntaan niihin määritellyn ehdon perusteella. Esimerkiksi ensimmäisen portin kohdalla varmistetaan, että hakijalla on oikeus asumistukeen. Hakijan oikeus asumistukeen on päätelty porttia edeltävässä sääntö-operaatiossa, joten arvon tulisi olla portin kohdalla boolean-tyyppinen *true* tai *false* eli tosi tai epätosi. Jos arvo on *false*, eli oikeutta asumistukeen ei ole, prosessi ohjataan päättymään. Jos arvo on *true* eli hakijalla on oikeus asumistukeen, jatketaan seuraavalle portille.

Seuraavaksi kiinnitetään huomiota laskentaprosessissa käytettäviin liiketoimintasääntöihin. Liiketoimintasääntöjä hyödynnetään aiemman toteutusesimerkin tapaan tilanteissa, joissa päätellään jonkin muuttujan arvo olemassa olevan tiedon perusteella. Sääntötiedostot aktivoituvat *Rule Taskeissa* eli sääntö-operaatioissa, kun niiden sääntöjoukkoa kutsutaan. Sääntömoottori käy säännöt läpi ja päättelee lopputulokset.

Liiketoimintasääntöjen toteutukseen hyödynnetään Excel-tiedostoja, joissa liiketoimintasäännöt järjestetään taulukkomuotoon (kuva 11).



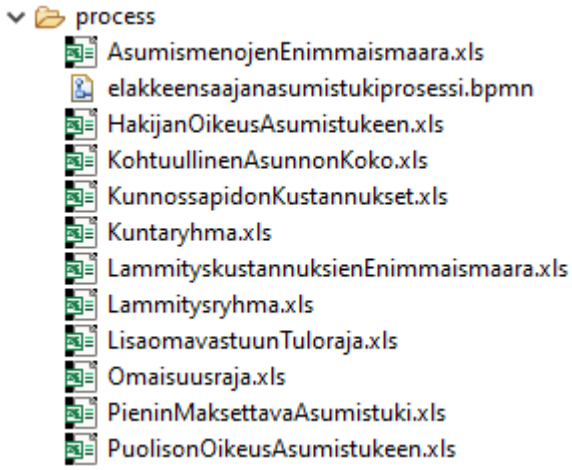
		Import	data.Puolso			
		Import	data.Ftuun			
		Import	data.Elakkeensaajanasumistukihakemus			
		Import	data.Elakkeensaajanasumistukitatkaisu			
			data.LammitysRyhma			
		RULEFLOW-GROUP	LammityskustannuksienEnimmaismaara			
		NO-LOOP	true			
		Notes	Asetetaan lämmityskustannuksien enimmäismäärä			
		RuleTable LammityskustannuksienEnimmaismaara				
		CONDITION	CONDITION	DATE-EFFECTIVE	DATE-EXPIRES	ACTION
		$hakemus: Elakkeensaajanasumistukihakemus				
		getAsunto().onValmistunutTaiPerusparannettuEnnen1974() == $param	getLammitysRyhma() != null && getLammitysRyhma().equals($param)			$hakemus.setLammityskustannuksienEnimmaismaara($param); update($hakemus);
	Lämmityskustannukset	Asunto on valmistunut tai perusparannettu ennen 1974	Lämmitysryhmä	PVM	PVM	Aseta lämmityskustannuksien enimmäismäärä
	Lämmitysryhmä 1 + ennen 1974	TRUE	LammitysRyhma.LAMMITYSRYHMA_1	01-JAN-2023	31-DEC-2023	2.70
	Lämmitysryhmä 2 + ennen 1974	TRUE	LammitysRyhma.LAMMITYSRYHMA_2	01-JAN-2023	31-DEC-2023	2.86
	Lämmitysryhmä 3 + ennen 1974	TRUE	LammitysRyhma.LAMMITYSRYHMA_3	01-JAN-2023	31-DEC-2023	3.06
	Lämmitysryhmä 1 + jälkeen 1974	FALSE	LammitysRyhma.LAMMITYSRYHMA_1	01-JAN-2023	31-DEC-2023	2.07
	Lämmitysryhmä 2 + jälkeen 1974	FALSE	LammitysRyhma.LAMMITYSRYHMA_2	01-JAN-2023	31-DEC-2023	2.20
	Lämmitysryhmä 3 + jälkeen 1974	FALSE	LammitysRyhma.LAMMITYSRYHMA_3	01-JAN-2023	31-DEC-2023	2.35

Kuva 11. Lämmityskustannuksien päättelyyn käytetty Excel-sääntötiedosto.

Kuvassa 11 on esitetty sama sääntöesimerkki kuin ohjelmointitoteutuksessa (luku 5.4), eli lämmityskustannuksien enimmäismäärän päättely. Excel-muotoisessa sääntötiedostossa jokainen rivi edustaa yhtä kokonaista liiketoimintasääntöä. Ensimmäisessä ehtolausekkeessa tarkistetaan, onko asunto valmistunut tai perusparannettu ennen vuotta 1974. Kun parametri vastaa riville kirjoitettua arvoa, siirrytään seuraavaan sarakkeeseen eli toiseen ehtolauseeseen. Toinen ehtolauseke tarkistaa asunnon lämmitysryhmän. Kaksi seuraavaa saraketta määrittelevät DATE-EFFECTIVE-ja DATE-EXPIRES-avainsanojen avulla ajanjakson, jolloin kyseinen sääntö on voimassa, eli alkaen 01.01.2023 ja päättyen 31.12.2023. Samaan sääntötiedostoon voidaan siis tarvittaessa lisätä usean eri ajanjakson sääntöjä, jotka aktivoituvat vain silloin, kun kyseinen ajanjakso on käynnissä. Viimeiseen sarake sisältää säännön

seurauksen, ja sarakkeeseen voidaan merkitä seurauslausekkeessa käytetty parametri.

Laskentaprosessin alkuperäisessä mallinnuksessa (kuva 8) liiketoimintasääntöjä hyödyntävät päättelyoperaatiot esitettiin violetilla värillä. Jokaisesta näistä operaatiosta muodostetaan Excel-sääntötiedosto, joka lisätään prosessin resurssikansioon (kuva 12). Resurssikansio sisältää myös itse prosessin (elakkeensaajanasumistukiprosessi.bpmn).



Kuva 12. Prosessissa käytetty resurssikansio.

Kun sääntötiedostot on saatu luotua, ne liitetään prosessin *Rule Taskeihin* niiden sääntöjoukon nimen (*Ruleflow Group)* avulla. Tällöin jokainen sääntöjoukko aktivoituu prosessin tietyssä vaiheessa. Vastaavasti laskurin metodikutsut liitetään ohjelmakoodina prosessin *Script Taskeihin*, jolloin laskuri suorittaa laskutoimitukset oikeassa vaiheessa.

Kuten ohjelmointitoteutuksessa, sovellus ajetaan testitapausten avulla (esimerkkikoodi 6).

DroolsTest.java

```
110
111     @Test
112     void testitapaus2() {
113
114         // Hakijan tiedot
115         hakija.setIka(67);
116         hakija.setEtuus(Etuus.VANHUUSELAKE);
117         hakija.setTulot(600);
118         hakija.setOmaisuus(1400);
119         hakija.setVelat(0);
120
121         // Liitetään hakija hakemukseen
122         hakemus.setHakija(hakija);
123
124         // Puolison tiedot
125         puoliso.setIka(65);
126         puoliso.setEtuus(Etuus.VANHUUSELAKE);
127         puoliso.setTulot(500);
128         puoliso.setOmaisuus(3200);
129         puoliso.setVelat(0);
130
131         // Liitetään puolison tiedot hakemukseen
132         hakemus.setPuoliso(puoliso);
133         hakemus.setHakijallaPuoliso(true);
134
135         // Asunnon tiedot
136         Asunto asunto = new Asunto();
137         asunto.setSijaintikunta(Kunta.HELSINKI);
138         asunto.setAsuntotyyppi(Asuntotyyppi.VUOKRA_ASUNTO);
139         asunto.setAsumismenot(770);
140
141         // Liitetään asunnon tiedot hakemukseen
142         hakemus.setAsunto(asunto);
143
144         // Liitetään hakemus ratkaisupohjaan
145         ratkaisu.setHakemus(hakemus);
146
147         // Data eli faktat syötetään sessioon
148         kieSession.insert(hakemus);
149         kieSession.insert(ratkaisu);
150
151         /*
152          * Hakemus ja ratkaisu merkitään myös session globaaleiksi muuttujiksi,
153          * jolloin niihin voidaan viitata Script Taskeista.
154          */
155         kieSession.setGlobal("hakemus", hakemus);
156         kieSession.setGlobal("ratkaisu", ratkaisu);
157
158         // Käynnistetään prosessi ja aktivoidaan kaikki säännöt
159         this.kieSession.startProcess("elakkeensaajanasumistukiprosessi");
160         kieSession.fireAllRules();
161         kieSession.dispose();
162
163         // Tulokset
164         assertEquals(303.11, ratkaisu.getMyonnetynTuenMaara());
165         assertEquals(true, ratkaisu.isTukiMyonnetty());
166     }
```

Esimerkkikoodi 6. Sääntömoottoritoteutuksen testi.

Laskentasovelluksen vaatimat lähtötiedot alustetaan samalla tavoin kuin ohjelmointitoteutuksessa. Tietojen syöttämisessä sovellukseen on kuitenkin eroa: hakemus- ja ratkaisu-muuttujat lisätään sääntömoottorin työmuistiin insert()-metodia käyttäen. Tämän jälkeen kyseiset muuttujat merkitään myös session globaaleiksi muuttujiksi, jotta niihin voidaan viitata laskentaprosessin eri vaiheissa. Lopuksi prosessi käynnistetään startProcess()-metodikutsulla, ja kaikki prosessissa olevat säännöt aktivoidaan fireAllRules()-metodikutsun avulla.

6 Toteutuksien vertailu

Kun molemmat toteutukset on saatu valmiiksi, voidaan ryhtyä tarkastelemaan työn tuloksia. Toteutuksien vertailussa kiinnitetään huomiota erityisesti laskentaprosessin ja liiketoimintasääntöjen implementointiin, toteutustapojen suorituskykyyn ja osaamisvaatimuksiin.

6.1 Sovellusten toiminnan vertailu

Koska laskenta oli kokonaisuudessaan melko yksinkertainen, prosessin muodostaminen oli hyvin suoraviivaista sekä sääntömoottorin kanssa että ilman sitä. Sääntömoottorin avulla liiketoimintaprosessi sai kuitenkin helpommin ymmärrettävän ja vaikuttavamman ulkonäön BPMN2-mallinnuksena. Lisäksi sääntömoottorin käyttämä mallinnus oli olennainen osa ohjelman suorittamista, kun taas ohjelmointitoteutuksessa prosessi ilmeni lähinnä vain abstraktilla tasolla. Laskennan suorittaminen tapahtui molemmissa toteutuksissa testitapausten kautta, ja prosessin kutsuminen ohjelmakoodista tapahtui molemmissa toteutuksissa samaan tyyliin.

Suurin ero toteutuksien välillä oli kuitenkin liiketoimintasääntöjen implementoinnissa. Ohjelmointitoteutuksessa säännöt sisällytettiin suoraan ohjelmakoodiin, kun taas sääntömoottorin avulla säännöt eristettiin ohjelmakoodista omaksi resurssipaketikseen.

Toisaalta myös suorituskyvyn puolesta toteutuksilla oli selkeä ero. Toteutuksia testattiin samalla testijoukolla, johon kuului neljä testitapausta. Kaikkien neljän testitapauksen suorittaminen vei perinteisessä ohjelmointitoteutuksessa keskimäärin 0,09 sekuntia, kun taas Droolsin avulla suorittaminen vei keskimäärin 1,9 sekuntia. Ero ei ole ajallisesti suuri, mutta koska sovellukset toteuttavat täsmälleen saman toiminnallisuuden, asia herättää huomiota. Yksi syy sääntömoottorin pidempään suoritusaikaan voi löytyä sääntöjen käsittelystä suorituksen alkaessa. Excel-taulukoissa olevat säännöt käännetään ensin DRL-

muotoisiksi ja siitä edelleen Java-koodiksi, jonka jälkeen ne kääntyvät muun lähdekoodin mukana tavukoodiksi.

6.2 Osaamisvaatimusten vertailu

Kun ohjelmistokehityksessä otetaan käyttöön uusi työkalu, täytyy huomioida uuden teknologian vaatimukset osaamisen suhteen. Sääntömoottorit eivät muodosta tässä suhteessa poikkeusta, mutta opinnäytetyön toteutusvaiheessa käytetyllä Drools-sääntömoottorilla on etunaan Java-ohjelmointikielellä toteutettu rakenne. Koska myös perinteinen ohjelmointitoteutus on toteutettu Java-kielellä, siirtyminen sääntömoottorin käyttöön tapahtui sulavasti. Tämän vuoksi myös uuden teknologian opetteluun käytetty aika pysyi kohtuullisena.

Sääntömoottorin sisällyttäminen mukaan sovelluksen toimintaan ei ollut kuitenkaan täysin ongelmatonta. Liiketoimintasääntöjen implementointi sääntömoottorin avulla tuotti haasteita erityisesti taulukkojen rakenteen vuoksi. Koska liiketoimintasääntöjen, eli myös säännöissä käytettävien Java-ohjelmakoodin pätkien, kirjoittaminen tapahtuu Excel-taulukossa, koodin täydennystä tai syntaksin korostuksen kaltaisia apu-ominaisuuksia ei ole saatavilla. Tästä johtuen sääntötiedosto ei osoita sisältämiään virheitä ennen kuin se ajetaan, ja sääntötiedostot jouduttiinkin testaamaan useaan otteeseen ajamalla sovellus. Sama haaste toistui myös prosessin *Script Taskeissa* eli ohjelmakoodia hyödyntävissä operaatioissa, joissa ei myöskään ole käytössä varsinaista koodieditointa, vaan tekstikenttä, johon suoritettava ohjelmakoodi sijoitetaan. Sovelluksen ja sen sisältämien liiketoimintasääntöjen *debuggausta* eli virheenselvitystä pystyttiin kuitenkin helpottamaan Log4j-lokituskirjaston avulla. Koska Drools kääntää .xls-muotoiset sääntötaulukot ensin DRL-sääntötiedostoiksi, pystyttiin tätä toiminnallisuutta hyödyntämään myös virheenselvityksessä. Taulukkomuotoiset sääntötiedostot voidaan tulostaa kehitysympäristön konsoliin DRL-muodossa (kuva 15), mikä helpottaa syntaksivirheiden havaitsemisessa.

```
package com.sample;
//generated from Decision Table
import data.Hakija;
import data.Asunto;
import data.Kunta;
import data.Kuntaryhma;
import data.ElakkeensaajanasumistukiHakemus;
import data.ElakkeensaajanasumistukiRatkaisu;
        no-loop true
        ruleflow-group "Omaisuusraja"
// rule values at C18, header at C13
rule "Omaisuusraja_18"
        date-effective "01-JAN-2023"
        date-expires "31-DEC-2023"
        when
                $hakemus: ElakkeensaajanasumistukiHakemus(isHakijallaPuoliso() == true)
        then
                $hakemus.setOmaisuusRaja(29290); update($hakemus);
end

// rule values at C19, header at C13
rule "Omaisuusraja_19"
        date-effective "01-JAN-2023"
        date-expires "31-DEC-2023"
        when
                $hakemus: ElakkeensaajanasumistukiHakemus(isHakijallaPuoliso() == false)
        then
                $hakemus.setOmaisuusRaja(18306); update($hakemus);
end
```

Kuva 13. Taulukosta DRL-muotoon käännettyjä sääntöjä.

Vaikka alustavan sääntötiedoston muotoilu oli työlästä, sen sijaan uusien sääntöjen lisääminen taulukoihin, sääntöjen muokkaaminen ja niiden lukeminen oli erittäin sujuvaa verrattuna perinteiseen ohjelmointiin. Myös kokonaisuutena arvioiden sääntömoottorin käytön haasteet keskittyivät sovelluksen alustavan version kehittämiseen, mutta järjestelmän ylläpito helpottui huomattavasti. Näihin havaintoihin perustuen liiketoimintasäännöstön ylläpito voitaisiin antaa tehtäväksi myös henkilölle, jolla ei ole ohjelmointikokemusta, mutta sääntötiedostojen rakenteen muodostaminen joudutaan edelleen uskomaan ohjelmistoasiantuntijan käsiin.

6.3 Toteutuksessa käytetyt työkalut

Laskentasovelluksen toteutukseen käytettiin Java-ohjelmointikielen versiota 8, joka on yhteensopiva Droolsin kanssa. Drools asennettiin käyttämällä Droolsin virallisilta verkkosivuilta ladattavaa asennuspakettia. Sovellus toteutettiin Eclipse-kehitysympäristössä, johon asennettiin kehityksen kannalta hyödylliset lisäosat Lombok ja BPMN2 Modeler. Lombokin tarkoituksena on vähentää

Boilerplate-koodin määrää erilaisten oikopolkujen avulla. Boilerplate-koodi tarkoittaa esimerkiksi Javan kaltaisen runsassanaisen ohjelmointikielen toistuvia rakenteita, joita ovat esimerkiksi luokkien get- ja set- metodit. BPMN2 Modeler on puolestaan nimensä mukaisesti BPMN-annotaatiota hyödyntävä mallinnusväline, jota voidaan hyödyntää esimerkiksi Droolsin kanssa. Toteutuksen apuvälineenä käytettiin myös Log4j-lokituskirjastoa, jonka avulla sovelluksen lokittamat tiedot olivat kattavampia ja tästä oli apua erityisesti virhetilanteiden selvityksessä. Projektin lähdekoodia ylläpidettiin Git-versionhallinnan avulla etärepositoriossa.

7 Yhteenveto

Opinnäytetyön tuloksena syntyneet toteutukset tarjoavat useita näkökulmia liiketoimintalogiikan toteuttamiseen sääntömoottorin avulla. Vaikka ilman sääntömoottoria toteutetun sovelluksen rakenteella pyrittiin jäljittelemään mallinnettua prosessia ja sääntömoottorin toimintaa, Droolsin avulla mallinnus integroitui suoraan sovelluksen toimintaan. Projektin lopputuloksena käy ilmi, että prosessin määrittelyn merkitys kasvaa liiketoimintamoottorin kanssa toimiessa, kun aiemmin mallinnettu prosessi luodaan uudelleen osaksi sovellusta.

Taulukkotiedostoihin sijoitetut liiketoimintasäännöt ovat helppolukuisia, ja sääntöjä voivat tulkita sekä ihmiset että itse sääntömoottori. Uusien taulukkomuotoisten liiketoimintasääntöjen lisääminen on helppoa, mikäli ne seuraavat samaa kaavaa kuin aiemmat säännöt. Myös niiden ylläpitäminen on helpompaa, kun säännöt on erotettu muusta ohjelmakoodista.

Opinnäytetyön toteutukseen valittu Drools on avoimen lähdekoodin projektina helposti lähestyttävä ja siihen liittyen löytyy paljon oppimismateriaalia ja dokumentaatiota. Droolsille on myös vaihtoehtoja, kuten esimerkiksi IBM:n kehittämä Operational Decision Manager (ODM), joka ulkoistaa sääntömoottorin toiminnan pilvipalveluksi ja mahdollistaa näin päätöksenteon palvelukutsujen avulla.

Lähteet

Aliverti, Esteban ja Salatino, Mauricio ja De Maio, Mariano. 2016. Mastering JBoss Drools 6.

Drools-dokumentaatio. Verkkoaineisto. <https://docs.jboss.org/drools/release/6.4.0.Final/drools-docs/html/>. Luettu 20.4.2022.

Eläkkeensaajan asumistuen määrän laskentaohje. Verkkoaineisto. <https://www.kela.fi/documents/20124/410194/elakkeensaajan-asumistuen-maaran-laskentaohje.pdf>. Luettu 14.4.2023.

Eläkkeensaajan asumistuki. Verkkoaineisto. <https://www.kela.fi/elakkeensaajan-asumistuki>. Luettu 4.4.2022.

Fiorini, Simone ja Gopalakrishnan, Arun V. 2015. Mastering jBPM6.

Frankenfield, Jake. Business Logic: Definition, Benefits, and Example. Verkkoaineisto. <https://www.investopedia.com/terms/b/businesslogic.asp>. 28.8.2020. Luettu 11.3.2023.

Gaikwad, Madhura. What is a Business Process? Definition, Examples, and Advantages. Verkkoaineisto. <https://blog.processology.net/what-is-a-business-process>. Luettu 24.4.2023.

Laki eläkkeensaajan asumistuesta (11.5.2007/571). Verkkoaineisto. <https://www.finlex.fi/fi/laki/ajantasa/2007/20070571>. Luettu 4.4.2022.

Liiketoimintasuunnitelma. Verkkoaineisto. <https://www.suomi.fi/yritykselle/yrityksen-perustaminen/yritystoiminnan-suunnittelu/opas/yritysideasta-liiketoiminnaksi/liiketoimintasuunnitelma>. 22.3.2019. Luettu 01.2.2023.

Martin, Matthew. Drools Tutorial: Drools Rule Engine Architecture & Examples. <https://www.guru99.com/drools-tutorial.html>. 8.4.2023. Luettu 20.4.2023.

Morris, Ben. 2016. What Do We Actually Mean When We Say Business Logic. Verkkoartikkeli. <https://www.ben-morris.com/what-do-we-actually-mean-when-we-say-business-logic/>. Luettu 11.3.2023.

Ross, Ronald G. ja Lam, Gladys S.W. 2016. Structured Business Vocabulary: Concept Models. Verkkoaineisto. <http://www.brcommunity.com/a2016/b880.html>. Luettu 2.2.2023.

Valtioneuvoston asetus eläkkeensaajan asumistuen määräytymisperusteista vuonna 2023. Verkkoaineisto. <https://www.finlex.fi/fi/laki/alkup/2022/20220977>. Luettu 1.2.2023.

What Are Business Rules?. Verkkoaineisto. <https://www.ibm.com/topics/business-rules>. Luettu 5.1.2023.

What Is Business Process Modeling?. 2021. Verkkoaineisto. <https://www.ibm.com/cloud/blog/business-process-modeling>. Luettu 5.4.2023.

What is low-code?. Verkkoaineisto. <https://www.ibm.com/topics/low-code>. Luettu 4.5.2022.

Laskentaesimerkki 1

Testitapaus 1. Hakija on 34-vuotias henkilö, joka ei saa eläkkeensaajan asumistukeen oikeuttavaa eläkettä tai etuutta. Testitapauksella ei lain mukaan ole oikeutta eläkkeensaajan asumistukeen.

Hakijan tiedot:
Ikä: 34
Etuudet: Ei ole
Tulot: 500 e/kk
Omaisuus: 500 e
Velat: 0 e
Onko hakijalla puoliso: Ei ole
Asunnon tiedot:
Sijaintikunta: Hyvinkää
Asunnon tyyppi: Vuokra-asunto
Asumismenot: 600 e/kk

Laskennan tulos:

- Hakijalla ei ole oikeutta eläkkeensaajan asumistukeen. Tukea ei voida myöntää.

Laskentaesimerkki 2

Testitapaus 2. Hakija on 67-vuotias henkilö, joka saa eläkkeensaajan asumistukeen oikeuttavaa vanhuuseläkettä. Hakijalla on lain mukainen oikeus eläkkeensaajan asumistukeen. Hakijalla on puoliso jolla on myös oikeus eläkkeensaajan asumistukeen.

Hakijan tiedot:
Ikä: 67
Etuus: Vanhuuseläke
Tulot: 600
Omaisuus: 1400
Velat: 0
Onko hakijalla puoliso: Kyllä
Puolison tiedot:
Ikä: 65
Etuus: Vanhuuseläke
Tulot: 500
Omaisuus: 3200
Velat: 0
Asunnon tiedot:
Sijaintikunta: Helsinki
Asunnon tyyppi: Vuokra-asunto
Asumismenot: 770 e/kk

Laskennan tulos:

- Sekä hakijalla että tämän puolisolla on oikeus eläkkeensaajan asumistukeen. Pariskunnalle voidaan myöntää yhteinen asumistuki, joka maksetaan kummallekin erikseen. Yhteisen asumistuen määrä on 606,237 e/kk, josta hakijalle maksettava osuus on puolet eli 303,11 e/kk.

Laskentaesimerkki 3

Testitapaus 3. Hakija on 52-vuotias henkilö, joka saa eläkkeensaajan asumistukeen oikeuttavaa takuueläkettä. Testitapauksella on myös oikeus eläkkeensaajan asumistukeen. Hakijalla on puoliso jolla ei ole oikeutta eläkkeensaajan asumistukeen.

Hakijan tiedot:
Ikä: 52
Etuus: Takuueläke
Tulot: 500
Omaisuus: 2000
Velat: 500
Onko hakijalla puoliso: Kyllä
Puolison tiedot:
Ikä: 57
Etuus: Ei ole
Tulot: 1500
Omaisuus: 3500
Velat: 1000
Asunnon tiedot:
Sijaintikunta: Jyväskylä
Asunnon tyyppi: Omistettu omakotitalo
Asumismenot: 8,33 e/kk
Omakotitalon pinta-ala: 120
Onko omakotitalo valmistunut tai perusparannettu ennen vuotta 1974: Kyllä

Laskennan tulos:

- Hakijalla on oikeus eläkkeensaajan asumistukeen, mutta maksettavaa etuutta ei voida muodostaa luovutettujen tietojen perusteella.

Laskentaesimerkki 4

Testitapaus 4. Hakija 45-vuotias henkilö, joka saa eläkkeensaajan asumistukeen oikeuttavaa työkyvyttömyyseläkettä. Testitapauksella on oikeus eläkkeensaajan asumistukeen. Hakijalla ei ole puolisoa.

Hakijan tiedot:
Ikä: 45
Etuus: Työkyvyttömyyseläke
Tulot: 400
Omaisuus: 5000
Velat: 1200
Onko hakijalla puoliso: Ei
Asunnon tiedot:
Sijaintikunta: Hamina
Asunnon tyyppi: Asumisoikeusasunto tai osaomistusasunto
Asumismenot: 520 e/kk
Laskennan tulos:

- Hakijalla on oikeus etuuteen. Hakijalle voidaan myöntää eläkkeensaajan asumistuki, jonka suuruus on 393,73 e/kk.