2011
2012

BEDRIJFSECONOMISCHE WETENSCHAPPEN

master in de toegepaste economische wetenschappen: handelsingenieur in de beleidsinformatica

Masterproef

Modelleren van processen en beslissingen : een geval studie

Promotor :
Prof. dr. Koenraad VANHOOF

Natasja Verdeyen

Masterproef voorgedragen tot het bekomen van de graad van master in de toegepaste economische wetenschappen: handelsingenieur in de beleidsinformatica

universiteit hasselt
KNOWLEDGE IN ACTION

Universiteit Hasselt | Campus Diepenbeek | Agoralaan Gebouw D | BE-3590 Diepenbeek
Universiteit Hasselt | Campus Hasselt | Martelarenlaan 42 | BE-3500 Hasselt

universiteit hasselt
KNOWLEDGE IN ACTION

2011
2012

BEDRIJFSECONOMISCHE WETENSCHAPPEN

master in de toegepaste economische wetenschappen: handelsingenieur in de beleidsinformatica

Masterproef

Modelleren van processen en beslissingen : een geval studie

Promotor :
Prof. dr. Koenraad VANHOOF

Natasja Verdeyen

Masterproef voorgedragen tot het bekomen van de graad van master in de toegepaste economische wetenschappen: handelsingenieur in de beleidsinformatica

universiteit hasselt
KNOWLEDGE IN ACTION

Inhoudsopgave

Samenvatting

Door de jaren heen zijn bedrijven hun processen steeds meer gaan automatiseren waardoor het steeds moeilijker werd een overzicht te hebben op alle processen binnen een bedrijf. Om deze reden is het modelleren van processen ontstaan maar ook deze procesmodellen werden op hun beurt steeds complexer. Dit is één van de redenen waarom men het Decision model heeft ontworpen. In de bedrijfsproces modellen zitten vaak heel wat beslissingen die men moet nemen. Aan deze beslissingen worden door de bedrijfsproces modellen een overbodige volgorde opgelegd. Vanuit deze idee wil men de bedrijfslogica die deze bedrijfsbeslissingen inhoudt, afzonderen van de bedrijfsprocessen. De bedrijfslogica wordt dan onafhankelijk van de bedrijfsprocessen gemodelleerd.

Het modelleren van de bedrijfslogica kan men ondermeer doen door middel van het Decision model. In dit onderzoek werd dit model dan ook grondig bekeken en onderzocht. De basiscomponenten van het Decision model zijn de Rule Family en het Decision diagram. De Rule Family oogt als een beslissingstabel maar is een bijzondere vorm van een beslissingstabel. Alvorens men een bepaalde beslissingstabel mag benoemen met de term Rule Family moet deze immers voldoen aan vijftien principes die uitvoerig worden beschreven in de theorie van het Decision model. Deze principes zorgen voor structurele eenvoud, een declaratief karakter en optimale integriteit. Men wil bekomen dat de Rule Families eenvoudig en ondubbelzinnig zijn, tevens wil men ervoor zorgen dat deze onafhankelijk zijn van bepaalde technologieën. De tweede basis component van het Decision model is het Decision diagram. Dit diagram geeft de relaties weer tussen de verschillende Rule Families. Deze relaties zijn ook zonder dit Decision diagram aanwezig tussen de Rule Families, al is de structuur hierin dan moeilijk terug te vinden. Het diagram zorgt er bovendien voor dat men bepaalde principes beter kan nagaan. Aan de hand van de case over het bedrijf Userv Financial Services, dat financiële diensten aanbiedt, wordt deze theorie toegepast. Uit onderzoek bleek dat de theorie zeer duidelijk toe te passen was. Er werden namelijk geen dubbelzinnigheden terug gevonden die problematisch bleken te zijn.

Zoals hierboven vermeldt, is de theorie van het Decision model onafhankelijk van elke technologie. Wanneer men echter, bij grote bedrijven, deze Rule Families

handmatig moet doorlopen, kost dit heel wat tijd. De bedrijfslogica wordt dan wel overzichtelijk met het Decision model, toch is het handig om deze Rule Families te implementeren in een programma dat de bedrijfsbeslissingen automatisch zal doen. Het Decision model is een zeer recente theorie; er zijn dus nog niet veel software producenten die applicaties ontworpen hebben om het Decision model te implementeren. Toch stijgt de interesse en intussen zijn er al enkele mogelijkheden op het internet te vinden. Twee mogelijkheden zijn de programma's Drools en OpenRules. Beide programma's worden hier verder uitgediept aan de hand van de implementatie van de Userv case.

Zowel Drools als OpenRules werken als plugin bij het programma Eclipse, een Java ontwikkelingsomgeving. Bij het onderzoeken of beide applicaties gebruiksvriendelijk en makkelijk toepasbaar zijn, is het meteen duidelijk dat men toch wel enige informatica achtergrond nodig heeft om met beide programma's te kunnen werken. Een goede kennis van de Java programmeertaal is een basisvereiste voor beide applicaties. Als men dan verder bekijkt hoe goed beide applicaties het doen op het vlak van de theorie over het Decision model, dan is het duidelijk dan men bij OpenRules specifiek mikt op het Decision model. Drools daarentegen, beoogt een meer algemene benadering van de beslissingstabellen. Bij Drools linkt men het Decision model niet rechtstreeks aan het programma, al biedt het wel alle mogelijkheden die vereist zijn om een Decision model te implementeren. Bij OpenRules daarentegen is dit wel het geval, men voorziet een leidraad waarin zelfs gerefereerd wordt naar de theorie over het Decision model.

Beide programma's worden getoetst aan de vijftien principes van het Decision model, waarbij Drools beduidend beter scoort op de inhoudelijke principes terwijl OpenRules beter presteert bij de visuele vereisten.

Toch kan er geen expliciete voorkeur uitgaan naar één van beide programma's, beide hebben immers sterke en zwakke punten. Het is aan het bedrijf dat gebruik wil maken van dergelijke programma's om te bekijken welke eigenschappen voor hen prioritair zijn.

Woord vooraf

Het schrijven van een masterproef is een zeer leerrijke ervaring maar vraagt ook heel wat inspanning. Daarom wil ik graag enkele mensen bedanken die mij hierin hebben bijgestaan.

Vooreerst wil ik graag mijn promotor Prof. dr. Koen Vanhoof bedanken voor het aanreiken van een interessante case en voor de begeleiding die hij mij heeft geboden. Mijn dank gaat eveneens uit naar Frank Vanhoenshoven, die mij ondersteuning heeft aangeboden bij het gebruiken van Drools. Eveneens apprecieer ik heel erg de hulp die ik kreeg van de OpenRules Support medewerkers.

Tot slot wil ik graag mijn familie en vrienden bedanken voor de steun die ik kreeg tijdens het verwezenlijken van deze masterproef.

Hoofdstuk I: De probleemstelling

Deze masterproef situeert zich binnen het domein business modelling, meer bepaald het modelleren van processen. Door de jaren heen zijn bedrijven hun processen meer en meer gaan automatiseren. Dit bleek een enorm succes en bracht een sterke bedrijfsgroei met zich mee. Heel wat ondernemingen stonden echter niet meer stil bij het managen van deze processen, wat vaak een totale chaos tot gevolg had. Men verloor de controle over de bedrijfslogica die vervat zat in de systemen.

Als reactie hierop startte men met het modelleren van de bedrijfsprocessen. Ook binnen de huidige bedrijfsrealiteit maakt men nog steeds gebruik van deze manier van werken. Voorheen werd alle logica in het procesmodel gestoken. Recent heeft men hiervoor een andere oplossing gevonden door gebruik te maken van de Decision Model theorie.

Deze theorie gaat er van uit dat men bij het modelleren de bedrijfslogica moet scheiden van de bedrijfsprocessen. De bedrijfsprocessen hebben een bepaalde volgorde en zijn dus procedureel, men spreekt hier over de "hoe" van een eenheid van werk. De bedrijfslogica, of ook wel bedrijfsbeslissingen genoemd, zijn daarentegen declaratief en hebben daarom geen specifieke volgorde. Hierbij spreekt men van de "wat" van een specifieke eenheid van werk. Als deze 'hoe' en 'wat' niet voldoende van elkaar worden onderscheiden, zal men een onnodige volgorde opdringen aan de bedrijfslogica. Bovendien heeft deze opsplitsing als voordeel dat het bedrijfsprocesmodel veel simpeler wordt en dat er een duidelijk overzicht is van de mogelijke combinaties van condities.

Wanneer de bedrijfslogica wordt gescheiden van de bedrijfsprocessen, zal men de bedrijfslogica kunnen aanpassen zonder dat men iets aan de bedrijfsprocessen hoeft aan te passen en ook omgekeerd is dit het geval. Bovendien zal het hergebruik van bedrijfproces modellen en Decision modellen door deze opsplitsing stijgen. (Goldberg, The Decision Model , A Business Logic Framework Linking Business and Technology, 2010)

Het Decision Model zorgt ervoor dat men conclusies kan trekken uit verschillende feiten die men in het bedrijf wil managen. Het model is daarmee niet zomaar een tekstuele opsomming van bedrijfslogica maar een gestructureerde uniforme

representatie van de bedrijfsbeslissingen. De theorie over het Decision model voorziet vijftien principes waaraan moet worden voldaan om een volwaardig Decision model te bekomen.

Deze theorie wil men uiteraard ook kunnen toepassen, daarom is het belangrijk te achterhalen of deze theorie toepasbaar is in de realiteit. Verder wil men ook graag weten of de theorie eenduidig en verstaanbaar is. Dit zal worden afgetoetst aan de hand van een case.

Na het opstellen van een dergelijk Decision model, kan men dit model benutten door het handmatig te doorlopen bij het nemen van beslissingen. Bij heel wat gegevens is dit te tijdrovend. Het zou daarom dan ook nuttig zijn deze Decision modellen op één of ander manier te implementeren zodat deze automatisch kunnen worden verwerkt. Het Decision model is nog een zeer recente theorie en net omwille van deze reden zijn er nog niet veel programma's die Decision modellen kunnen implementeren. Drools en OpenRules zijn twee programma's die deze mogelijkheid zouden kunnen bieden. In dit onderzoek zal nagegaan worden in hoeverre men een Decision model kan implementeren in deze programma's en bijgevolg zo zijn beslissingen kan automatiseren met deze programma's. Bovenop de mate waarin de programma's implementeerbaar zijn, is het ook zinvol te weten of de programma's gebruiksvriendelijk zijn en wat de eventuele voor- of nadelen bij gebruik kunnen zijn, om tot slot aan bedrijven te kunnen aangeven welk van deze twee programma's de voorkeur krijgt.

Hoofdstuk II : The Decision Model

De theorie die in deze masterproef wordt toegepast, is deze van het Decision Model. In dit hoofdstuk wordt deze theorie eerst verder toegelicht.

A. Wat is The Decision Model?

Het Decision Model is een sjabloon waarmee men de bedrijfslogica achter de bedrijfsbeslissingen wil organiseren en beheren.

Met het Decision Model wil men geen fysiek model bekomen waarmee men aangeeft hoe de logica moet worden geïmplementeerd in een bepaalde technologie. Het is namelijk zo opgemaakt dat het volledig onafhankelijk is van een bepaalde technologie. Het model gaat er van uit dat de bedrijfslogica op zichzelf bestaat, onafhankelijk van hoe het wordt uitgevoerd of waar in het bedrijf het wordt uitgevoerd. Het model heeft een herkenbare structuur waarbij drie doelen voor ogen worden gehouden:

- Eenvoudig om te beheren en te interpreteren
- Declaratief zodat het eveneens onafhankelijk is van technologie
- De bedrijfslogica moet consistent zijn met zichzelf en met bedrijfsdoelstellingen

De bedrijfslogica waarvan voorheen reeds sprake was, kan worden gedefinieerd als conclusies die een bedrijf bekomt door de analyse van verschillende feiten. Een voorbeeld hiervan luidt : " Een verkoper die voor een waarde van meer dan 10 000 euro aan goederen verkoopt per maand, krijgt een bonus van 50 euro op het maandloon." In dit voorbeeld is het feit; dat de verkoper goederen voor een waarde van meer dan 10 000 euro verkoopt. Als conclusie zal deze verkoper een bonus van 50 euro op het maandloon krijgen. Dit is een zeer eenvoudig voorbeeld maar het geeft duidelijk weer wat er wordt bedoeld met een feit en een conclusie.

Het Decision Model heeft verschillende principes. Aan de hand van deze principes wil het model ervoor zorgen dat de conclusies en feiten atomair, precies, eenduidig en overeenkomstig met de bedrijfsdoelstellingen worden weergegeven.

Het basiselement waaruit het Decision Model bestaat is een tweedimensionale structuur, waarin aan de ene kant de feiten en aan de andere kant de conclusie

worden weergegeven. Men bekomt hierdoor een beslissingstabel. Een beslissingstabel is een tabel met alle mogelijke gebeurtenissen en de acties die hiervoor ondernomen moeten worden (Worldwebonline "Decision table"). Het Decision model benoemt deze tweedimensionale tabel als Rule Family, wat niet hetzelfde is als een beslissingstabel. Een Rule Family en een beslissingstabel verschillen namelijk op twee manieren.

- Om een Rule Family te bekomen moet men aan een hele set van principes voldoen. (deze principes worden later in detail besproken)
- Een Rule Family kan gerelateerd zijn aan één of meer andere Rule Families. Deze relaties moeten ook aan enkele eenduidige principes voldoen. Beslissingstabellen zoals deze gedefinieerd werden hierboven hebben daarentegen geen onderlinge relaties.

Hieruit blijkt duidelijk dat men deze twee benamingen niet door mekaar mag gebruiken, er is een duidelijk verschil tussen beide.

Het tweede element waaruit het Decision Model bestaat is het Decision Model diagram. Dit diagram geeft de structuur van het Decision Model weer. De gedetailleerd versie van de Rule Families worden hierin niet getoond, alleen de onderlinge relaties tussen deze Rule Families worden weergegeven. Voor het opstellen van een Decision Model diagram hoort men eveneens een aantal regels te volgen. Deze zullen later aan bod komen.

a. Impact van bedrijfsbeslissingen op een onderneming

Eén van de belangrijke redenen van het ontstaan van het Decision Model is het belang van bedrijfsbeslissingen binnen een onderneming. De impact van een bedrijfsbeslissing kan worden omschreven door middel van een functie van drie karakteristieken. Elk van deze drie karakteristieken handelt over een ander aspect van de bedrijfsbeslissing en ze verduidelijken ook waarom het Decision Model zo nuttig is. Deze drie karakteristieken zijn de operatieve context, de grootte van de economische impact en de complexiteit van de bedrijfslogica. In wat volgt zal op elke karakteristiek dieper worden ingegaan om zo de invloed op het Decision Model te verduidelijken.

i De operatieve context

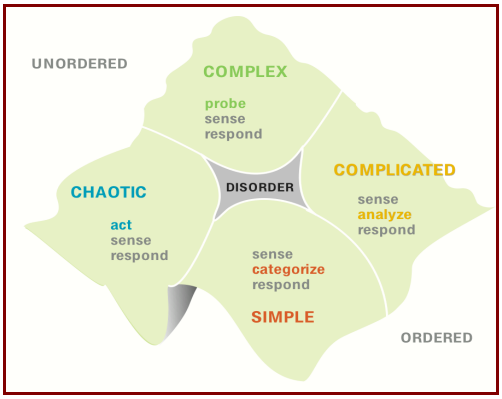
Voorheen werd reeds aangehaald dat de conclusie van een bedrijfsbeslissing afhankelijk is van feiten. Toch is dit niet in alle gevallen zo, er bestaan immers gevallen waarbij feiten niet aanwezig zijn of dat deze niet eenduidig zijn. Er zijn verschillende redenen die tot deze afwezigheid kunnen leiden, zoals een mogelijke crisis of andere onvoorziene omstandigheden, wat wil zeggen dat men beslissingen zal moeten nemen zonder dat de nodige feiten expliciet gekend zijn. Binnen een bedrijf zal men daarom een beroep moeten doen op de expertise en de intuïtie van de personen die beslissingen moeten nemen. In zo een geval wordt er gesproken over patroon of gebeurtenis gebaseerde beslissingen.

De operatieve context wordt verder verduidelijkt in "The Cynefin Framework", die afgebeeld wordt in figuur 1. Dit is een model dat aangeeft hoe bedrijfsleiders op een goede manier beslissingen kunnen nemen. In het model worden vier verschillende operatieve contexten onderscheiden:

- *De eenvoudige operatieve context*; hierbij weet men welke informatie men nodig heeft en is deze ook beschikbaar voor diegene die de beslissing moet nemen. Zoals in de figuur wordt aangegeven, moet men informatie eerst verzamelen, vervolgens categoriseren (eventueel door middel van het Decision model) om tot slot beslissingen te kunnen nemen. Om terug te komen op het Decision model wil dit zeggen dat in deze context alles aanwezig is om de nodige Rule Families op te stellen. Zowel de kolomhoofding, als de condities, als de conclusies zijn gekend. In deze eenvoudige operatieve context is het Decision model erg nuttig en geeft het een klare kijk op de logica achter de bedrijfsbeslissingen.
- *De gecompliceerde operatieve context* bevindt zich net zoals de eenvoudige operatieve context in het geordende domein. Hiermee wordt bedoeld dat men in deze context weet welke informatie men nodig heeft maar heeft men deze informatie niet ter beschikking. Volgens "The Cynefin framework" moet men alle beschikbare data verzamelen om vervolgens analyses uit te voeren zodat uiteindelijk beslissingen kunnen worden genomen. Concreet voor het Decision model wil dit zeggen dat de feit types (kolomhoofdingen) gekend zijn maar dat de gegevens voor het invullen van de condities en de conclusies onvolledig of niet aanwezig zijn. In dit geval is het Decision Model eveneens van nut, het zal

er immers voor zorgen dat men weet welke informatie ontbreekt. Op deze manier is het mogelijk te achterhalen welke expertise een bedrijf nodig heeft om tot de nodige informatie te komen. Daarnaast is er ook steeds de mogelijkheid dat de bekomen informatie niet eenduidig is, ook hiervoor is het Decision model nuttig. Door de verschillende data te gebruiken om de Rule Families op te maken, zal men immers al de mogelijke alternatieven tegen mekaar kunnen afwegen. Door een correcte analyse van deze alternatieven kan men tot een juiste beslissing komen.

- *De complexe operatieve context* behoort in tegenstelling tot de twee vorige contexten niet tot het geordende domein maar wel tot het ongeordende domein. In deze context weet men niet welke informatie men exact nodig heeft waardoor men uiteraard ook niet in staat is deze informatie te verzamelen. Het is in deze context dan ook erg moeilijk om een Decision model op te stellen. Eerst en vooral moet men onderzoeken wat men wil weten en welke informatie men hiervoor nodig heeft. Het Decision model zou hier echter wel van nut kunnen zijn om te helpen bij het zoeken naar mogelijke oplossingen. Op deze manier is het mogelijk om een complexe context te herleiden tot een minder gecompliceerde of zelfs tot een eenvoudige context.
- Tot slot is er *de chaotische operatieve context,* die zich ook in het ongeordende domein bevindt. In deze context is er alleen chaos en moeten er eerst stappen worden ondernomen om enige orde in de wanorde te scheppen alvorens men beslissingen kan nemen. Het Decision model kan hierbij helpen doordat men kleine stukjes van de chaos verzamelt en hier opnieuw orde in vindt. Zo kan men stap per stap dichter bij een complexe context komen.



Figuur 1 : The Cynefin Framework (Snowden & Boone, 2007)

ii De grootte van de economische impact

Bedrijfsbeslissingen kunnen op basis van verschillende eigenschappen worden ingedeeld. Eén hiervan is het indelen op basis van het aantal keer dat een beslissing moet worden gemaakt per dag, per week, per jaar, enz. In het algemeen zijn er twee soorten beslissingen die binnen een bedrijf kunnen worden genomen. De operationele beslissingen zijn beslissingen die vaak in grote aantallen worden genomen en vaak terugkomen. Daarnaast zijn er de strategische beslissingen, die minder frequent moeten worden genomen en dus maar in kleine hoeveelheden voorkomen. Het aantal keren dat een bepaalde beslissing wordt genomen, staat vaak in schril contrast met de economische waarde van die beslissing. Eén enkele operationele beslissing heeft een veel kleinere impact of economische waarden dan één strategische beslissing. Daarom moeten we de totale economische impact van een beslissing zien als het product van de economische waarde van één enkele beslissing en het aantal keren dat deze beslissing binnen een bepaalde tijdseenheid wordt genomen.

Vaak wordt er minder aandacht besteed aan operationele beslissingen omdat hun individuele economische waarden erg klein zijn ten opzichte van strategische beslissingen. Wanneer men echter de totale economische waarde van deze operationele beslissingen bekijkt zoals hierboven beschreven, dan blijkt hieruit dat deze beslissingen vaak ook van groot belang zijn voor een onderneming. Het is zinvol om ook deze beslissingen trachten te automatiseren en hierbij kan het Decision model zeker van dienst zijn. Door de controle over deze operationele beslissingen te bewaren kan men immers strategische voordelen behalen ten opzichte van concurrenten.

iii De complexiteit van de bedrijfslogica

Hoe groter een bedrijf is, hoe ingewikkelder de beslissingen vaak zijn. Dit geldt ook voor het Decision model; naarmate er meer Rule Families zijn en deze ook meer feit types bevatten zal het Decision model complexer worden. Er kan een opdeling gemaakt worden in drie categorieën van complexiteit. Voor elk van deze categorieën kan het Decision model van waarde zijn.

- De minste complexiteit is terug te vinden wanneer er maar een klein aantal Rule Families zijn en deze Rule Families elk maar ongeveer een vijftal feit types

bevatten. In zo een geval zorgt het Decision model voor standaardisering en een goede communicatie.

- Vervolgens is er de gemiddelde complexiteit, waarbij er meer Rule Families zijn dan bij de minste complexiteit maar deze hoeveelheid blijft nog onder de vijftig en ook het aantal feit types overstijgt de tien niet. Het Decision model kan hier voor duidelijkheid en standaardisatie zorgen en men kan er ook mee bepalen welke beslissingen moeten worden geautomatiseerd en welke best door mensen worden genomen.
- Als laatste is er het hoogste niveau van complexiteit waarbij men vijftig, honderd of zelfs duizend Rule Families kan hebben die ieder meer dan tien feit types bevatten. Op dit niveau kan het toepassen van het Decision model een concurrentieel voordeel bieden, het kan er immers voor zorgen dat men een duidelijker overzicht krijgt van de bedrijfslogica.

Het verenigen van deze karakteristieken.

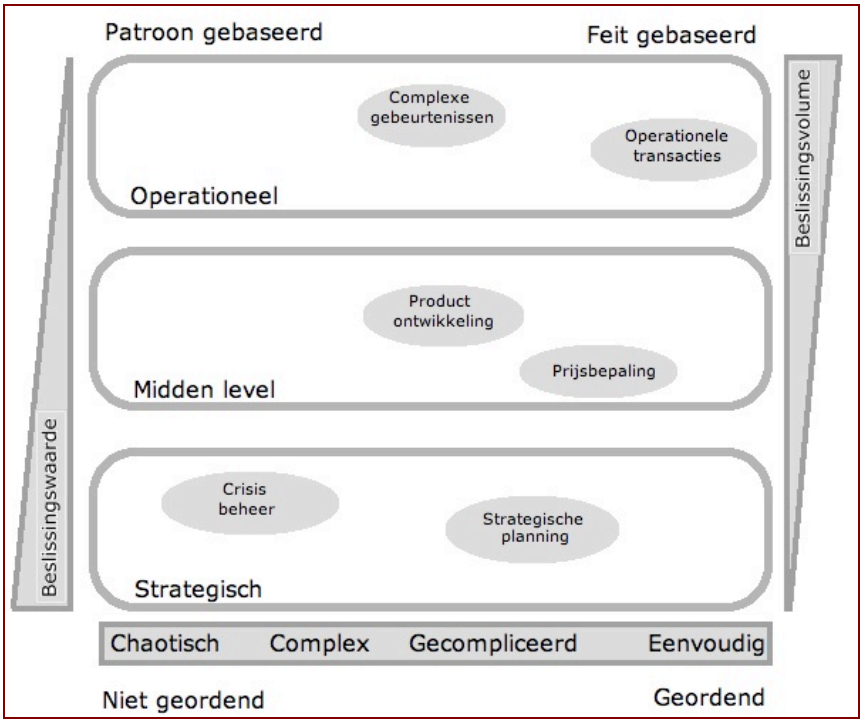
Het Decision model kan voor elke van deze karakteristieken afzonderlijk worden bekeken en ook de waarde van het model voor elk van deze karakteristieken kan men duidelijk benoemen. In realiteit heeft men uiteraard steeds het samenspel van deze karakteristieken, wat niet altijd zo eenvoudig te vatten is. Om hier toch een duidelijker beeld van te krijgen, worden alle karakteristieken en hun onderlinge relaties weergegeven in figuur 2. Bij wijze van voorbeeld worden er in de grijze ellipsen enkele bedrijfsbelissingstypes weergegeven, zo wordt het visueel duidelijk welke karakteristieken deze hebben.

Eén van deze bedrijfsbelissingstypes is "Crisis beheer", deze ellips bevindt zich helemaal links onder. Dit is een strategische beslissing en heeft bijgevolg een grote economisch waarde maar slechts een klein volume van dit soort beslissingen is in een organisatie aanwezig. Bovenaan de figuur wordt weergegeven dat het om een patroon gebaseerde beslissing gaat en onderaan kan men vaststellen dat dit eerder een chaotische context is in een ongeordend domein.

Een tegenhanger van bovenstaand voorbeeld is de ellips waar "Operationele transacties" in vermeld staat. Deze bevindt zich rechts boven in het operationele vak. Dit bedrijfsbeslissingstype heeft een kleine individuele economische waarde maar wordt in een groot volume uitgevoerd binnen een organisatie. Het bevindt

zich in het geordende domein en heeft een eenvoudige context. Verder valt af te lezen van de figuur dat het gaat om een feiten gebaseerde beslissing.

Bovenstaande voorbeelden worden verduidelijkt door de onderstaande figuur waarop de relaties tussen de verschillende karakteristieken van bedrijfsbeslissingen worden weergegeven.



Figuur 2 : Categoriseren van bedrijfsbeslissingen

B. Voordelen van het Decision Model

Waar in het vorige deel het Decision model duidelijk werd gedefinieerd, zal in wat volgt aandacht worden besteed aan de meerwaarde die het model biedt.

Doordat men de Rules Families zo eenvoudig en eenduidig wil maken, brengt dit ook met zich mee dat men wijzigingen snel en eenvoudig kan aanbrengen. Eerst en vooral is de bedrijfslogica onafhankelijk van de bedrijfsprocessen waardoor een van beide kan worden aangepast zonder dat de andere hier hinder van ondervindt. Een ander voordeel dat hieruit voortvloeit is dat men veranderingen in de bedrijfslogica zeer eenvoudig kan aanpassen. Wanneer bepaalde conclusies of feiten veranderen hoeft men deze immers slechts aan te passen in de Rule Family waarvan ze deel

uitmaken, andere Rule Families zullen hier geen hinder van ondervinden. Dit is een erg belangrijk voordeel, aangezien in de huidige bedrijfscontext alles sneller en steeds sneller verandert. Het is dan ook van belang dat deze veranderingen zo snel mogelijk kunnen worden aangebracht. Zo kan men beslissingen nemen die rekening houden met alle feiten, ook met de laatste nieuwe.

The Decision model zorgt er steeds voor dat de complexiteit tot een minimum gereduceerd wordt. Men kan dit eigenlijk veralgemenen naar eender welk model, als het een goed model is zal de complexiteit geminimaliseerd worden zodat men een duidelijk overzicht krijg van wat men heeft gemodelleerd.

Het Decision model is onafhankelijk van elke technologie, wat in de actuele maatschappij een erg belangrijke meerwaarde is. De markt heeft heel wat mogelijkheden te bieden op het vlak van technologie en iedere onderneming heeft hier zijn voorkeuren.

C. Het opstellen van het Decision model

a. Voorbereiding

Als een bedrijf beslist om gebruik te maken van het Decision model komt hier meer bij kijken dan alleen het opstellen van het model volgens de bijhorende principes. Aan het opstellen gaan een aantal andere overwegingen vooraf die men zeker niet over het hoofd mag zien om tot een correct resultaat te komen.

Vooreerst moet het management een grondige analyse maken van wat ze willen bereiken. Men maakt een duidelijk overzicht van de algemene doelen die men voor ogen heeft, om van daaruit deze doelen meer te concretiseren en hier eventueel cijfers op te plakken. Hierbij bekijkt men ook op welke termijn men deze doelen wil realiseren en welk budget hiervoor beschikbaar is. Als dit overzicht volledig is, wordt een project team samen gesteld dat zich dan kan bezighouden met het opstellen van het Decision model. Een dergelijk team kan bestaan uit een project manager, een bedrijfsgovernance manager, een bedrijfsgovernance raad, bedrijfsanalisten, een kennismanager, bedrijfsexperten, IT professionelen, enz.

Dit project team zal vervolgens het bereik van het project bepalen, net zoals dit bij andere projecten gedaan wordt. Hierbij zijn er zes aspecten waarvan men het bereik moet bepalen.

- Het bedrijfsproces model, waardoor men kan bepalen waar in het model het Decision model van nut zal zijn.
- Als tweede aspect bepaalt men het bereik van de beslissingen. Dit doet men door het bepalen van de complexiteit van de bedrijfslogica, de operatieve context, de economische waarde, …
- Bepalen van de transacties die de grootste invloed hebben op een eindbeslissing.
- Bepalen van feit types die later zullen kunnen worden gebruikt bij het opstellen van het Decision model.
- Het bepalen van een maatstaf waarmee men het Decision model kan afwegen ten opzicht van de doelen die men heeft vooropgesteld.
- Een schatting maken van de grootte van het Decision model. Hierbij maakt men een schatting van het aantal Rule Families, rijen in de Rule Families, enz.

Tot slot stelt men een gedetailleerd bedrijfsproces model op zodat men een overzicht krijgt van welke processen extra uitwerking nodig hebben. Deze verdere uitwerking wordt gedaan door middel van het Decision model.

Na deze voorbereidingsfase, is de volgende stap het opstellen van het Decision model zelf. Dit verloopt volgens verschillende principes, die in wat volgt volledig worden verduidelijkt.

Eenmaal de Decision modellen klaar zijn, moeten deze uiteraard worden getest alvorens men deze kan implementeren. Tot slot kan men de Decision modellen automatiseren door het gebruik van software tools.

b. De principes voor het opstellen van een Decision model

Het Decision model heeft drie sterke kwaliteiten. Deze zijn structurele eenvoud, declaratief karakter en optimale integriteit. Om deze drie kwaliteiten te waarborgen in een goed Decision Model moet men onderstaande principes in acht nemen.

i De eerste zeven principes

De eerste zeven principes hebben structurele eenvoud als gemeenschappelijk doel. Deze principes zorgen er voornamelijk voor dat het Decision model eenvoudig is, makkelijk te begrijpen en slecht op één manier te interpreteren.

Principe 1 : Het tabel principe

Dit eerste principe gaat over de algemene vorm van het Decision model. Het geeft de basis weer van hoe een Decision model er moet uitzien.

De fundamentele structuur is de Rule Family, deze heeft twee dimensies. Eén hiervan is de hoofding (heading), de andere is het midden gedeelte (body). De basis van het Decision model bestaat dus uit een twee dimensionale tabel.

Principe 2 : Het hoofding principe

Het volgende principe geeft aan dat de hoofding bestaat uit een set van feit types.

Een feit is een stuk informatie en een feit type is een algemene classificatie van een feit. Vb feit = 5 jaar, feit type = leeftijd. De feiten zelf staan niet in de titel, alleen de feit types vindt men daar terug.

Er bestaan zowel complexe als eenvoudige feit types. Het maakt evenwel niet uit welk type er wordt gebruikt maar het is wel belangrijk dat er een duidelijke bedrijfsdefinitie is voor elk feit type en dat er eenduidige beperkingen worden gelegd op het domein van ieder feit type.

Principe 3 : Het cel principe.

De inhoud van een cel is steeds een atomaire logische uitdrukking die conform is met de hoofding. Deze atomaire logische uitdrukking bestaat steeds uit een operator + operand. Voorbeelden van operatoren zijn " is minder dan ", " is in ", " is tussen ",... enz. Deze moeten uiteraard samen worden bekeken met de feit types, er moet namelijk een duidelijke samenhang zijn tussen het feit type en de gebruikte operator. Zo kan men bijvoorbeeld niet zeggen dat een straatnaam groter dan een bepaald cijfer moet zijn. Dit kan daarentegen wel voor een

huisnummer. Het domein dat men in principe twee heeft bepaald, is bepalend voor de operand. Deze moet namelijk binnen dit domein liggen. Als het domein bijvoorbeeld "datum" is kan de operand " 10/10/10" zijn maar niet "123/1234/12345"

Principe 4 : Het rij principe

De rijen van een Rule Family bestaan uit condities die leiden tot een conclusie. Ieder feit type in de hoofding speelt een rol in een conditie of een conclusie, de onderverdeling tussen beide noemt men de feit type rol. Op deze manier bindt men de verschillende cellen in de Rule Family aan mekaar. Dit gebeurt door middel van functionele afhankelijkheid. Functionele afhankelijkheid betekent dat een waarde uit een cel of een groep van cellen op een unieke wijze een andere cel zal bepalen.

Principe 5 : Het conclusie principe

Een Rule Family kan maar één conclusie hebben, er is dus maar één conclusie kolom. Een rij met meer dan één conclusie feit type, is eigenlijk een geneste structuur en is dus niet atomair. Dit wil zeggen dat er evenveel Rule Families zijn als conclusie feit types. Het creëren van aparte twee dimensionale tabellen per conclusie feit type kan vaak leiden tot heel wat vragen, zoals bijvoorbeeld : "Leidt de ene conclusie tot de andere of zijn het toch de conditie feit types die tot deze conclusie leiden?". Dit vijfde principe zorgt er eigenlijk voor dat er meer Rule Families zijn. Dit lijkt op het eerste gezicht misschien ingewikkelder maar toch is dit niet het geval; de eenvoud van dit principe berust zich op het feit dat men de bedrijfslogica maar op één enkele manier kan representeren en interpreteren.

Als aan deze vijf principes voldaan is zal de Rule Family er uit zien zoals in tabel 1. Het is evenwel mogelijk dat er meer conditie kolommen zijn of dat er meer rijen voorkomen in de tabel.

Tabel 1 : Voorbeeld Rule Family

	Conditie				Conclusie	
Rule Pattern	Feit type		Feit type		Feit type	
1	Operator	Operand	Operator	Operand	Is	Operand
1	Operator	Operand	Operator	Operand	Is	Operand
2	Operator	Operand			Is	Operand

Principe 6: Het conditie principe

Al de condities in de Rule Family moeten waar zijn vooraleer de conclusie kan waar zijn. Het is dus een AND relatie tussen de condities en geen OR relatie. Dit voorkomt geneste structuren in de condities. Hier komen we ook een nieuw begrip tegen, namelijk Rule Patterns. Dit is een set van Rule Family rijen met een overeenkomstige set van conditie cellen die worden ingevuld. Principe 6 bevat vier subprincipes:

- Elke Rule Family moet minstens één Rule Pattern hebben. Een Rule Pattern wordt vaak pas ontdekt als men de Rule Family vervolledigt.
- De conditiesleutel van een Rule Pattern kan leeg zijn als er maar één Rule Pattern is in de Rule Family. Een conditiesleutel is een set van feit types in een Rule Pattern die de ingevulde conditie cellen bepaalt. Bij het opmaken van een Decision model spreken we over lege cellen en niet over nul waarden als er niet ingevulde cellen zijn. Er zijn twee mogelijkheden dit zich kunnen voordoen waardoor er lege cellen ontstaan. De waarde van de cel kan irrelevant zijn voor de rest van de rij of de waarde van de cel kan ongekend zijn.
- Een conditiesleutel van een Rule Pattern kan niet gedeeltelijk leeg zijn tenzij deze hele conditiesleutel leeg is.
- Een conclusie in een Rule Pattern mag niet afhankelijk zijn van een partiële sleutel.

Principe 7 : Het connectie principe

Een Rule Family heeft een relatie met een andere Rule Family als het conclusie feit type van één van beide een conditie feit type is in de andere Rule Family. Dit principe heeft ook twee subprincipes:

- Een Rule Family kan geen lege conclusie cellen hebben. Want dit is de "lijm" die het geheel samen houdt.
- Een Rule Pattern kan geen lege conclusie cellen hebben, dit volgt uit bovenstaand subprincipe.

De volgorde in de relaties tussen de Rule Families maakt geen deel uit van het Decision Model, wel van het proces model.

Eerste en tweede normaalvorm

Het Decision model introduceert drie normaal vormen waarvan de twee eerste in dit deel behandeld worden.

De eerste normaalvorm, waaraan voldaan is als principes 5 en 6 gelden, legt op dat een Rule Family slechts op één manier kan worden geïnterpreteerd. Deze normaalvorm leidt tot eenvoud bij het interpreteren en gemakkelijkheid bij het aanpassen van het model. Geneste structuren worden vermeden in de conclusie door principe 5 en in de condities door principe 6.

De tweede normaalvorm geeft aan dat partiële afhankelijkheid niet mogelijk is. Concreet wordt hier bedoeld dat een conclusie in een Rule Pattern niet afhankelijk mag zijn van een partiële conditiesleutel. Het gaat hier dus om cellen die niet leeg zijn maar toch geen invloed hebben op de conclusie cel. De oplossing is dus om deze te verwijderen.

ii Principe acht tot en met tien

Zoals voorheen vermeld moet het Decision model declaratief zijn, zodat het onafhankelijk is van een bepaalde technologie. Het mag dus niet procedureel zijn, wat wil zeggen dat het geen volgorde aangeeft. Hieruit volgen principes acht tot en met tien.

Principe 8: Het declaratieve hoofding principe

De feit types in de hoofding van de Rule Families mogen geen specifieke volgorde hebben. Dit wil zeggen dat de volgorde van de kolommen in een Rule Family geen betekenis heeft.

Principe 9 : Het declaratieve midden principe

De rijen in de Rule Family hebben geen specifieke volgorde, er zit geen betekenis achter de volgorde waarin deze rijen worden weergegeven.

Principe 10 : Het declaratieve relatie principe

Er zit geen volgorde in de manier waarop Rule Families met elkaar verbonden worden. Het Decision model diagram (dit wordt verder in deze tekst uitgelegd) waarin deze relaties worden weergegeven, kan men van boven naar onder, van links naar rechts of omgekeerd lezen. De manier hoe men dit leest heeft geen achterliggende betekenis. Over het algemeen zijn er twee manieren om het Decision Model te lezen, Forward-Chaining en Backward-Chaining.

Bij de eerste mogelijkheid start men bij de feiten die waar zijn om vervolgens al deze conclusies op te slaan, waarna men opnieuw nagaat of er andere condities waar geworden zijn, enzovoort.

Bij Backward-Chaining daarentegen start men met een conclusie en gaat men vervolgens na of de condities voor deze conclusie zijn voldaan. Beide manieren zijn goed voor het Decision Model, er is geen voorkeur voor één van beide. Al is Forward-Chaining meestal de manier waarvoor een persoon intuïtief zal kiezen.

iii De laatste vijf principes

De laatste groep principes behandelen de optimale integriteit van het Decision model. Men wil er met deze principes voor zorgen dat de inhoud van het Decision model zinnig is op het vlak van structuur, logica en bovenal ook bedrijfsgewijs zinvol. Met structuur wordt er bedoeld dat de redundantie geminimaliseerd moet worden. Logica wil zeggen dat de bedrijfslogica consistent en compleet moet zijn. Tot slot bedoelt men met bedrijfsgewijs, dat het de bedrijfsperformantie op die manier beïnvloedt zoals men tot doel had.

Principe 11: Rule Pattern transitieve condities principe

Dit principe wil streven naar een minimale redundantie. Er mogen in de condities van Rule Patterns geen afhankelijkheden zijn om een bepaalde conclusie te bekomen.

De derde normaalvorm

Door het toepassen van principe 11 staat het Decision Model in de derde normaalvorm, waarmee functionele afhankelijkheden tussen condities worden geëlimineerd. Concreet wil dit zeggen dat er in geen enkele rij een conditie is die leidt tot een conclusie die gaat over een andere conditie. Als een Rule Family in de derde normaalvorm staat mag men deze niet meer kunnen opsplitsen in meerdere conclusies.

Principe 12: Rule Family en Rule Pattern consistentie principe

Een Rule Family moet vrij zijn van inconsistenties binnen Rule Patterns en tussen Rule Patterns. Dit is zeker geen eenvoudig principe wat ondermeer blijkt uit een opbouw van zeven subprincipes.

- Het uitvoeren van een Rule Pattern moet resulteren in één conclusie of geen conclusie, deze kan niet resulteren in meerdere conclusies.

- De condities van een Rule Pattern moeten alleen de deelverzameling van waarden van een conditie feit type domein bevatten die binnen het bereik van de input gegevens liggen.
- Binnen een Rule Pattern mag er geen overlap zijn tussen de condities, dit kan namelijk tot meerdere conclusies leiden.
- Er mag geen overlap zijn tussen de condities van twee of meer verschillende Rule Patterns van een Rule Family.
- Een Rule Family moet tot ten minste één conclusie leiden. Dit wil zeggen dat een bepaalde Rule Pattern niet tot een conclusie kan leiden maar dat er minstens één andere Rule Pattern binnen een Rule Family wel steeds in een conclusie moet resulteren. Uiteraard is dit alleen het geval wanneer er een geldige invoer is.
- Een Rule Family kan resulteren in meerdere conclusies bij een invoer van een set geldige gegevens. Dan moeten deze conclusies echter steeds tot een andere Rule Pattern behoren, omdat binnen één Rule Pattern slecht één conclusie mag worden bekomen. Een bedrijf moet zelf uitmaken of het wenselijk is dat een Rule Family meerdere conclusies kan hebben.
- De conclusies van een Rule Family moeten alleen die waarden van het conclusie feit type domein bevatten die men als bedrijf verwacht in de conclusies.

Principe 13: Rule Family transitieve condities principe

Er mogen geen transitieve afhankelijkheden zijn tussen Rule Families van één Decision model. Een transitieve afhankelijkheid is aanwezig wanneer een conclusie feit type in één Rule Family dienst doet als een conditie feit type in twee andere Rule Families en het conclusie feit type van één van deze twee laatste Rule Families dienst doet als conditie feit type in de andere. Dit kan men makkelijker terugvinden als men het Decision model diagram dat hieronder wordt beschreven, bekijkt. Deze transitieve afhankelijkheden worden namelijk weergegeven als lussen. In zeer grote Decision models zijn deze lussen uiteraard moeilijk terug te vinden.

Principe 14 : Inferentie integriteit principe

Elke conclusie in een Rule Family die moet dienen als een conditie van een andere Rule Family, moet ook echt een Rule Family hebben waar deze conditie in terug te vinden is.

Principe 15 : Bedrijfssamenhang principe

Het Decision model moet bedrijfsdoelen trachten te behalen die men aan de hand van bepaalde waarden kan managen.

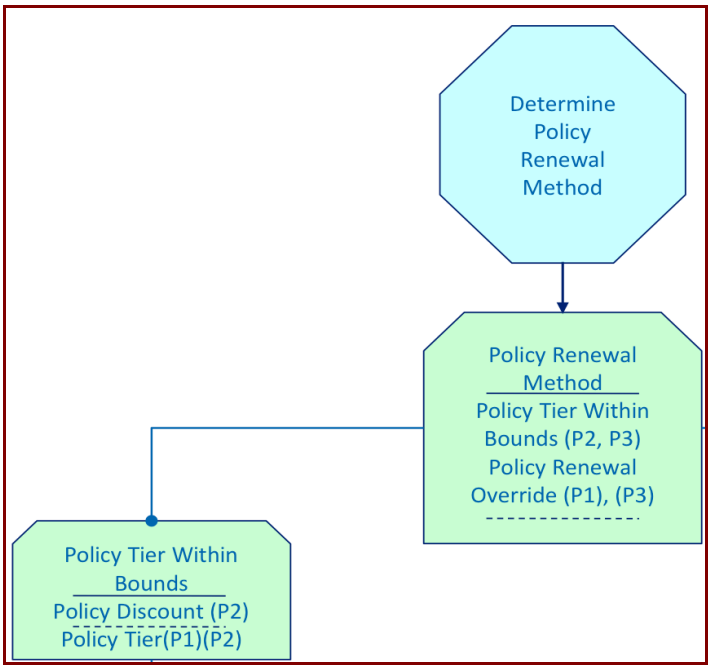
c. Het Decision model diagram

Om het geheel van Rule Famillies en hun onderlinge relaties beter te kunnen begrijpen en een duidelijk overzicht te hebben, stelt men deze voor in een Decision model diagram. In figuur 3 wordt een voorbeeld van zo een Decision model diagram weergegeven. Het Decision model diagram is samengesteld uit vier verschillende vormen die ieder één van de basis elementen van het Decision model voorstellen.

- De achthoek representeert de volledige bedrijfsbeslissing die men wil nemen. In de achthoek wordt steeds vermeld welke bedrijfsbeslissing men wil bekomen.
- Vervolgens is er de Rule Family vorm, deze wordt weergegeven als een rechthoek met twee stompe hoeken bovenaan, ook wel een "tombstone" genoemd. Boven de volle lijn wordt de naam van de Rule Family weergegeven, dit is ook de naam van het conclusie feit type. Boven de stippellijn komen de conditie feit types die een sleutel vormen tussen deze en een andere Rule Family. Dit wil zeggen dat deze conditie feit types in een andere Rule Family voorkomen als conclusie feit types. De conditie feit types die men aanbrengt als data zonder dat deze als conclusie feit type dienen in een andere Rule Family worden weergegeven onder de stippenlijn. Met de "P" naast elke conditie feit type wordt aangeduid in welke Rule Patterns het conditie feit type voorkomt.
- De bedrijfsbeslissing vorm wordt steeds verbonden met de Rule Family vorm door middel van een volle lijn die aan de Rule Family eindigt in een pijl.
- Twee Rule Families worden verbonden door middel van een volle lijn. Aan de kant van de Rule Family, waarvan het conclusie feit type dient als conditie feit type in de andere Rule Familie wordt de lijn beëindigd met een punt.

Bij voorkeur snijden de lijnen zich niet, zo blijft het overzicht over het volledige Decision diagram behouden. Indien het Decision diagram zeer groot en onduidelijk wordt kan men dit best opsplitsen in meerdere aparte Decision diagrammen. Men maakt dan best een Decision diagram per bedrijfsbeslissing. Kortom men zou er

best voor zorgen dat het Decision diagram zo overzichtelijk mogelijk is, zodat het snel en makkelijk te lezen is.



Figuur 3 : Voorbeeld van een Decision model diagram

Hoofdstuk III : Toepassen van de theorie op de UServ case

A. De Userv case

De case die door Prof. dr. Koen Vanhoof werd aangereikt gaat over Userv Financial Services. Userv is een onderneming die een heel portfolio aan financiële producten aanbiedt en wil zo goed mogelijk aan de behoeften van zijn klanten tegemoet komen. Hierbij willen ze zowel de bedrijven als gezinnen bereiken. Uiteraard is UServ een bedrijf dat zijn winst wil maximaliseren, daarom moeten ze een afweging maken tussen de diensten die ze hun klanten aanbieden en de risico's die hieraan verbonden zijn. Om dit te verwezenlijken heeft de onderneming enkele basis bedrijfsregels opgesteld, die de kern vormen van het uitvoerende beleid. Deze bedrijfsregels gaan over verkiesbaarheid, prijsbepaling en annuleringsmaatregelen en deze gelden zowel op individueel niveau als op portfolio niveau.

Zoals voorheen vermeld biedt UServ een erg uitgebreid gamma aan financiële producten aan zijn klanten aan. In deze case zal echter slechts een subset van deze producten worden behandeld, namelijk het onderdeel voertuigverzekeringen. Al de bedrijfsregels werden in een overzichtelijk document aangereikt. De regels bepalen of iemand in aanmerking komt voor een verzekering en welke premie er voor de eigenaar en de auto moeten betaald worden.

B. De toepasbaarheid van het Decision model op de Userv case.

Voor het opstellen van het Decision model ,toepasbaar op Userv case, worden de in het eerste hoofdstuk vermelde regels gevolgd. In wat volgt wordt beoordeeld in hoeverre de theorie toepasbaar is op de Userv case. Hierbij zal worden bekeken of de theorie duidelijk geformuleerd is, zodat deze kan worden toegepast zonder verdere opzoekingen van bepaalde termen of theorieën. Daarnaast zal eveneens worden gelet of de eenduidigheid van de Decision Model theorie. "Zijn bepaalde principes voor interpretatie vatbaar? Zit er dubbelzinnigheid in de principes? " zijn de hoofdvragen bij dit onderdeel. Tot slot wordt er ook aandacht besteed aan de moeilijkheidsgraad van het toepassen, namelijk :in hoeverre is bijkomende kennis vereist of heeft men automatisering nodig om bepaalde vereisten op een

realistische manier na te gaan. Gelet op het feit dat de Userv case geen extreem moeilijke structuren bevat en slechts een subgroep van Userv zijn totaal aantal producten bevat, kan deze case als relatief eenvoudig worden beschouwd. Voor moeilijkere cases dan deze zou het Decision Model ook toepasbaar moeten zijn.

In het onderdeel "voorbereiding" dat in het vorige hoofdstuk werd besproken, vindt men een uiteenzetting van taken die een bedrijf moet uitvoeren alvorens men van start kan gaan met het opstellen van het eigenlijke Decision Model. Aangezien deze case aangereikt werd met alle nodige informatie werd de voorbereiding reeds door Userv zelf gedaan en beperkt deze verhandeling zich slechts tot het opstellen van het Decision model. Hiervoor zullen de principes stap voor stap worden doorlopen zodat aan elk van de vijftien principes voldaan is en het Decision model in derde normaal vorm staat.

Principe 1

Principe één is de basis van een Rule Family, namelijk de tweedimensionale tabel. Uit praktische overwegingen werd ervoor geopteerd deze tabellen weer te geven aan de hand van het programma Excel, dit programma biedt alle grafische mogelijkheden die nodig zijn om de Rule Families weer te geven.

Om dit principe toe te passen heeft men geen extra voorkennis nodig. Als men niet met Excel kan werken kan men de tabel immers ook handmatig neerschrijven, maar uiteraard is dit erg omslachtig. Er rijzen ook geen verder vragen bij het toepassen van dit principe.

Principe 2

Als tweede stap wordt er aan de slag gegaan met de case zelf. De case werd zeer grondig doorgenomen om er alle feit types uit te kunnen destilleren. Vervolgens heb ik bij elk feit type ook al de mogelijke waarden van het feit type weergegeven.

Dit zijn enkele voorbeelden, de volledige lijst is terug te vinden in bijlage 1.

- Convertible car : "Yes" , "No"
- on " High Theft Probability Auto " list : "Yes", "No"
- Potential theft rating :" High", "Moderate", "Low"
- Make and model : "VW Bug"," Porsche"," BMW"
- Auto eligibility : "Not eligible"," Provisional"," Eligible"

Dit principe geeft eveneens geen moeilijkheden of onduidelijkheden. Als de voorbereidingsfase immers goed is verlopen, kan men een duidelijk overzicht verwachten van de verschillende feit types.

Principe 3

Voor elke waarde van een feit type werd vervolgens een operator uitgezocht die voldeed aan de bedrijfsregels die in de UServ case worden vermeld. Hierbij werd hierbij rekening gehouden met het feit dat deze operator ook logisch moet zijn ten opzichte van de waarde van het feit type en het feit type zelf. Onderstaand voorbeeld komt uit de UServ case: het feit type is " Vehicle Policy eligibility score" en kan alle cijferwaarden aannemen beginnende van 0 tot 250 en groter. Als operatoren wordt gebruik gemaakt van "Is Less Then", "Is Between", "Is greater than" en "Is", dit zijn operatoren die zich zeer goed lenen tot het gebruik met cijfers.

Tabel 2 : Voorbeeld van operator en operand

Vehicle Policy eligibility score	
Is Less Than	100
Is between	(100, 251)
Is greater than	250
Is	any

Net als bij de voorgaande principes lijken ook hier geen problemen te rijzen. Er kruipt evenwel wat tijd in om per feit type een geschikte operator te vinden vermits dit iets is wat manueel moet gebeuren. Eenmaal echter de geschikte operatoren gevonden zijn, hoeft men deze niet meer aan te passen, tenzij er veranderingen zijn in de beslissingen.

Principe 4

In elke Rule Family die werd opgemaakt, behoort elk feit type tot een conclusie of tot de condities.

Principe 5

Elke Rule Family van het Decision model heeft slechts één conclusie kolom, de Rule Families die twee conclusie kolommen bevatten, werden opgesplitst, zodat ook aan dit principe werd voldaan.

Principe 6

Het eerste deel van dit principe was relatief makkelijk te verwezenlijken, al de combinaties van condities komen immers tot een conclusie door middel van een AND operator.

De subprincipes zijn al wat tijdrovender, alle Rule Families hebben minstens één Rule Pattern, dit nakijken was vrij snel gedaan en dit geldt ook voor het tweede subprincipe. Voor het derde en het vierde subprincipe is wat meer waakzaamheid noodzakelijk. Eerst en vooral worden alle conditiesleutels van elke Rule Pattern bepaald om vervolgens na te gaan of deze aan de principes voldeden. Dit principe is niet moeilijk en ook erg duidelijk, maar vraagt wel enige tijd.

Principe 7

Aan hand van dit principe kwamen de relaties tussen de Rule Families tot stand. Hier is geen extra werk aan verbonden en dus ook geen extra moeilijkheid. Belangrijk hierbij is om op te merken dat deze relaties er alleen zijn als het conclusie feit type hetzelfde is als het conditie feit type. Men moet dus waakzaam zijn op het dubbel gebruik van feit types zodat men geen onnodige relaties tot stand brengt, of het omgekeerde, dat men hetzelfde feit type gebruikt waar nodig, anders zal de relatie er niet zijn. Ondanks het feit dat er in de theorie van het Decision model geen aandacht aan wordt besteed, is dit zeker niet evident wanneer men met veel Tule Families te maken krijgt. In zulke gevallen is het overzicht vaak zoek. Op dit punt kan een Decision Diagram, dat later in dit hoofdstuk besproken wordt, van nut zijn.

De subprincipes daarentegen zijn makkelijk te na te gaan.

Eerste en tweede normaalvorm

Door het toepassen van principe vijf en zes staan de Rule Families van Userv in tweede normaalvorm, hiervoor moeten geen extra principes toegepast worden.

Principe 8

Het toepassen van dit principe 8 is opnieuw relatief eenvoudig: in elke Rule Family wordt nagegaan of de volgorde van de kolommen kan worden veranderd zonder dat dit effect heeft op de beslissing. Het is hier dan vooral belangrijk te letten op verborgen volgordes, het kan namelijk lastig zijn om deze terug te vinden.

Principe 9

Het nagaan van dit principe is vrij makkelijk en bevat verder ook geen dubbelzinnigheden.

Principe 10

Als men dit principe bekijkt lijkt dit op het eerste zicht erg eenvoudig. Intuïtief zou men al makkelijk gaan denken dat de onderlinge relaties tussen de Rule Families toch een volgorde opleggen. Dit is echter niet het geval, wat erg misleidend kan zijn. Net daarom is er bij dit principe opnieuw enige oplettendheid nodig. Als men dit principe niet toepast zoals het hoort, kan men zonder het te beseffen toch een volgorde opleggen aan de Rule Families. Om na te gaan of dit niet gebeurd was bij de Userv Case werden enkele voorbeeld data gebruikt en vervolgens zowel Forward- als Backward-Chaining toegepast. Zo kon tot de conclusie gekomen worden dat er geen volgorde zit in de relaties tussen de Rule Families.

Principe 11

Eerst en vooral moet men elke combinatie van condities bekijken om na te gaan of er afhankelijkheden zijn. Dit moet men manueel doen en vraagt erg veel tijd. Als men afhankelijkheden vindt in een bepaalde Rule Family is het belangrijk na te gaan of dit toeval is of niet. Dit kan men alleen achterhalen door het aan een bedrijfsexpert te vragen. Vermits de mogelijkheid niet bestond om bij personen uit het bedrijf navraag te doen, werd uitgegaan van de veronderstelling dat er geen afhankelijkheden aanwezig zijn.

De derde normaalvorm

Het Decision model voldoet aan principe 11 , waardoor het ook in de derde normaalvorm staat.

Principe 12

Het is niet eenvoudig om na te gaan of er geen inconsistentie is in de Rule Patterns en tussen de Rule Pattterns. Hiervoor kan men gebruik maken van gesofisticeerde software maar omwille van praktische beperkingen werd in dit geval het principe echter manueel bekeken. De verschillende subprincipes van principe 12 hebben ervoor gezorgd dat dit iets makkelijker werd.

- Het eerste subprincipe is eenvoudig na te gaan en bleek de formulering eenduidig te zijn.

- De verwachte waarden voor een bepaald feit type werden reeds in bijlage 1 weergegeven. Alleen deze waarden werden opgenomen in de Rule Family. Als men per feit type zo een overzicht voorziet is dit principe makkelijk toe te passen en zeer eenduidig.
- Als men de waarden in tabel 1 bekijkt en nagaat of deze niet overlappen, is ook aan dit principe voldaan.
- Bij dit principe wil men voorkomen dat éénzelfde input voor twee verschillende conclusies kan zorgen. Om dit na te gaan moeten de Rule Patterns van elke Rule Family grondig worden bekeken. Dit subprincipe op zich is eenduidig en duidelijk maar de toepassing vergt enige tijd.
- De gevalideerde input bestaat uit de mogelijk feit type waarden die in bijlage 1 worden weergegeven. Met deze waarden worden al de Rule Families getest, om zo na te gaan of die steeds tot minstens één conclusie komen. Ook dit kan worden geautomatiseerd, maar omdat Userv nog een redelijk beperkte case is werd dit hier manueel gedaan. Het is een makkelijk maar tijdrovende bezigheid.
- Dit subprincipe hoeft geen extra aandacht, het is meer informatief.
- Het laatste subprincipe wordt eveneens door tabel 1 duidelijk. In deze tabel staan alle mogelijk waarden voor de conclusie feit types. Buiten deze hoeven er geen conclusie feit typewaardes te worden weergegeven in de Rule Families.

Principe 13

Dit is een niet te onderschatten principe, de transitieve afhankelijkheden tussen Rule Families zijn niet zo makkelijk terug te vinden. Eerst en vooral werd getracht deze transitieve afhankelijkheden terug te vinden door de relaties tussen de Rule Families uit te zoeken. Op die manier konden geen fouten tegen dit principe worden vastgesteld in de Userv case. Na het opstellen van het Decision diagram (zie bijlage 3) werd echter duidelijk dat er toch een lus was. Het vaststellen van deze fout bleek dan misschien wel eenvoudig met het Decision diagram maar als men grotere Decision diagrammen heeft met meer Rule Families zal dit steeds moeilijker worden. Het principe is dus bijgevolg wel eenduidig en duidelijk maar het testen wordt steeds moeilijker naarmate het Decision diagram groter wordt.

Een oplossing bieden voor de fouten tegen dit principe is niet evident. Meestal is de oplossing het verwijderen van een conditie feit type uit een Rule Family,

aangezien deze conditie al in een andere Rule Family getest werd waardoor reeds aan deze conditie voldaan is. Om te achterhalen welk conditie feit type men moet verwijderen is toch enige oplettendheid aangewezen.

Principe 14

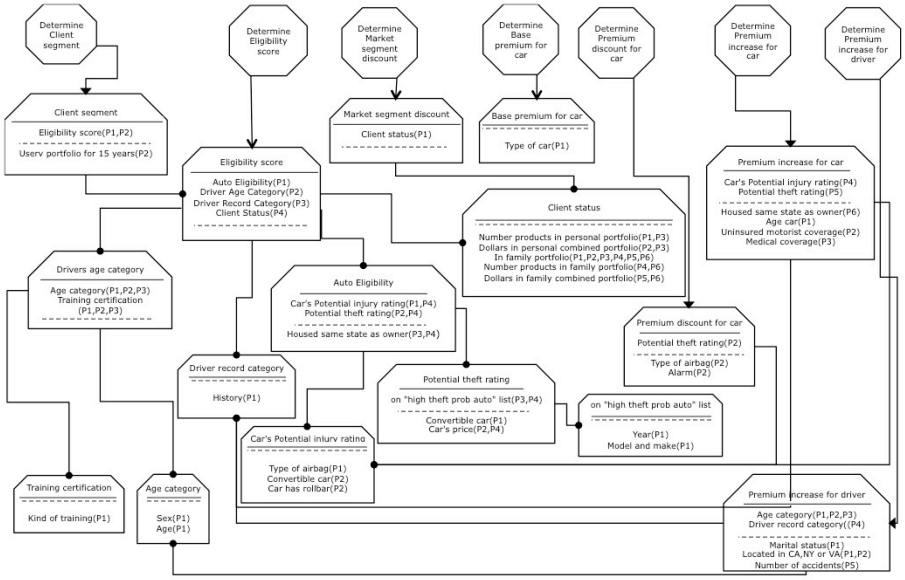
Bij dit principe moet worden nagegaan of de conclusie feit type waarden van een Rule Family die in tabel 1 werden weergegeven ook allemaal terug komen in de condities van de Rule Family die door voorgaan Rule Family worden ondersteund. Dit is vrij makkelijk na te gaan.

Principe 15

Dit laatste principe is erg moeilijk om na te gaan, aangezien er niemand deelneemt aan dit onderzoek van het bedrijf Userv, is er niet exact geweten wat hun doelen zijn. Men kan alleen terugvallen op de case. Als het Decision model wordt bekeken samen met de case lijkt dat alle doelen die worden gesteld ook effectief worden gehaald. Dit laatste principe is evenwel voor interpretatie vatbaar. Het bedrijf moet zelf beslissen of dit is wat men tot doel had. Allicht kan de ene persoon hier een andere mening over hebben dan de andere persoon. Zoals in de theorie wordt gesteld, is dit dus inderdaad één van de belangrijkste en moeilijkste principes.

Het Decision diagram

Voor het opstellen van het Decision diagram zijn de richtlijnen zeer duidelijk. Als men alle Rule Families heeft opgesteld, kan men aan de hand van deze richtlijnen een goed Decision diagram opstellen. Hoewel er geen dubbelzinnigheden in deze richtlijnen zitten, kan men wanneer men een zeer groot aantal Rule Families heeft best overwegen om het Decision diagram op te delen in verschillen delen. In figuur 4 vindt men het Decision diagram voor de Userv case. Zoals eerder aangehaald is dit een vrij beperkte case maar kan men toch al een redelijk kluwen van verbindingen waarnemen. Een meer leesbare versie is terug te vinden in bijlage 3. Dit Decision diagram werd opgesteld in Word maar voor grotere Decision diagrammen is gepaste software aangeraden. Als men het Decision diagram zonder grafische software wil realiseren neemt dit wel enige tijd in beslag, men moet niet alleen alle relaties weergeven maar men moet er ook voor zorgen dat het Decision diagram leesbaar is. Als alle verbindingen elkaar kruisen wordt het een onoverzichtelijke warboel.



figuur 4 : Het Decision diagram van de Userv case

Algemene beoordeling.

De vijftien principes voor het opstellen van een Decision model zijn allemaal erg eenduidig, als men deze wil toepassen stoot men niet op dubbelzinnigheden die tot verschillende resultaten kunnen leiden. De principes zijn duidelijk geformuleerd waardoor men geen extra achtergrondkennis nodig heeft om deze te begrijpen. Enige oplettendheid is echter wel vereist, bij het toepassen van de principes mag men de details niet over het hoofd zien.

Het grootste struikelblok is misschien wel de te investeren tijd. Als men al deze principes correct wil toepassen, neemt dit heel wat tijd in beslag. Deze tijdsduur zal stijgen naarmate men meer beslissingen wil opnemen in het Decision model. Sommige van deze principes kan men toepassen met behulp van speciale software, dit is dan ook aan te raden aangezien arbeidsuren toch steeds erg kostelijk zijn. Bovendien kan men het risico dat personen menselijke fouten maken niet helemaal uitsluiten. Software daarentegen doet exact datgene waarvoor het geprogrammeerd is.

Als besluit kan men stellen dat de theorie over het Decision model sluitend is en indien men er voldoende tijd in wil investeren, ook makkelijk toepasbaar.

Hoofdstuk IV : Implementatie in Drools

A. Wat is Drools?

Als vertrekpunt werd de website van Drools grondig bekeken. Op deze manier werd duidelijk welke mogelijkheden de software te bieden heeft en hoe men hiermee aan de slag kan gaan. Drools is een open source programma en men kan het gratis afhalen van de website. (Newton, et al.)

Het is een Business Logic Integration Platform en is een geïntegreerd platform voor Rules, Workflows en Event Processing. Er zijn verschillende projecten van Drools:

- Drools Guvnor : Business Rules Manager
- Drools Expert : Rule Engine
- Drools Fusion : event processing en temporal reasoning
- Drools Planner: automated planning

Dit zijn community versies van JBoss.org en hebben bijgevolg geen ondersteuning.

In deze casestudie werd enkel gebruik gemaakt van Drools Expert.

Alvorens verder in te gaan op het gebruik van Drools, is het belangrijk het verband tussen Drools en het Decision model aan te stippen. Het Decision model heeft tot doel bedrijfsbeslissingen te groeperen om er zo voor te zorgen dat deze overzichtelijker worden, men deze makkelijk kan communiceren en begrijpen. Daarnaast wil men ook het nemen van de beslissingen zelf vergemakkelijken. Het Decision model geeft aan welke beslissing men moet nemen aan de hand van bepaalde invoer data. Als men steeds manueel deze Rule Families moet doorlopen op zoek naar de juiste beslissingen, zal men allicht niet veel voordeel halen uit het Decision model. Dit is namelijk een zeer tijdrovende bezigheid. Bovendien kan en mag men ook de mogelijkheid van het maken van een menselijke fout niet verwaarlozen. Dit is dan ook meteen de reden waarom men Drools kan gebruiken. Al de Rules Families die men heeft opgesteld aan de hand van de theorie van het Decision model kan men implementeren in Drools. Hoe men deze Rule Families exact moet implementeren zal later in dit hoofdstuk nog aan bod komen. Als al de Rule Families geïmplementeerd zijn, zal men data kunnen ingeven aan de hand

waarvan Drools een beslissing aanbiedt die volgt uit de logica van het Decision model. Drools zal dus het nemen van beslissingen aan de hand van het Decision model automatiseren. In wat volgt zal worden onderzocht of Drools makkelijk te hanteren is. Vervolgens zal worden nagegaan of deze automatisering ook voldoet aan alle principes van het Decision model. Daaropvolgend wordt stilgestaan bij wat de eventuele struikelblokken en moeilijkheden zijn bij het implementeren van de Userv case in Drools.

B. Hoe verloopt de eerste kennismaking met Drools?

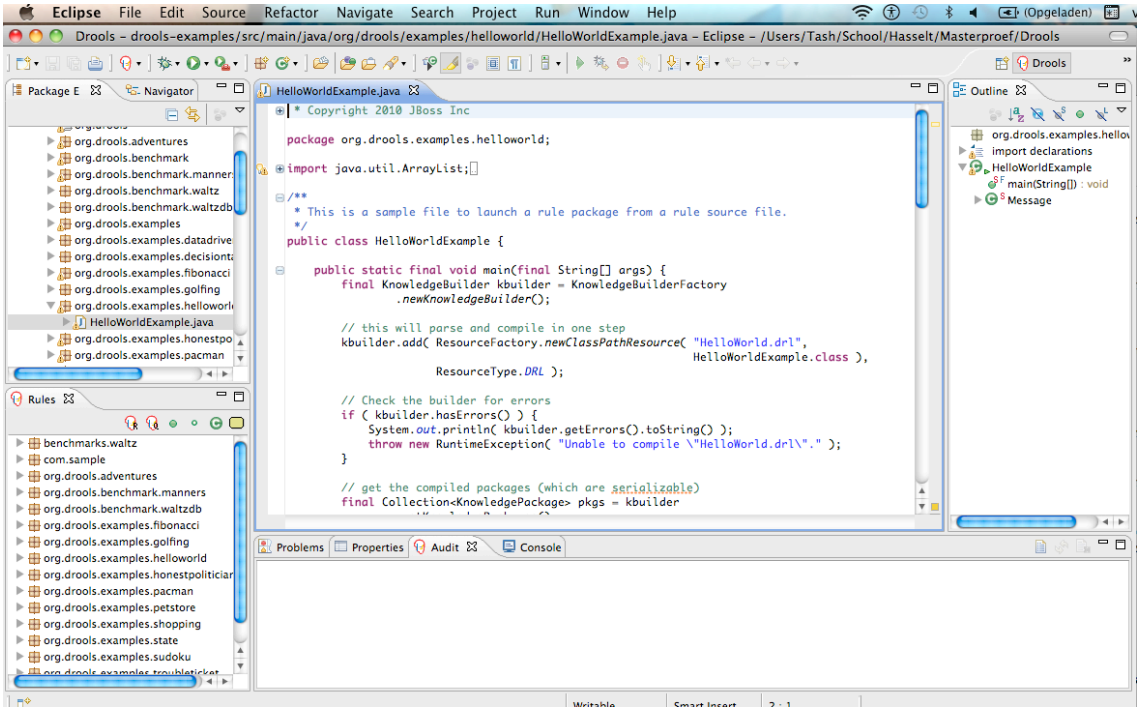
Zoals reeds eerder vermeld kan men Drools gratis afhalen. In deze casestudie werd dan ook gebruik gemaakt van zo een gratis versie.

Na het lezen van de handleiding over de introductie van Drools werd duidelijk dat ook het programma Eclipse noodzakelijk is om met Drools te kunnen werken. (The Eclipse Foundation, 2012) Eclipse is een java omgeving waarin men Drools kan gebruiken als plugin om de Rules aan te maken in Eclipse. Men voegt door Drools een Rule engine toe aan Eclipse. Dit kan men zien als een "black box" waarin men de Rules stopt om er dan de conclusies als resultaat weer uit te krijgen.

De installatie van Eclipse verliep probleemloos, voor deze studie werd de installatie gedaan op een computer met OS Mac en hierop geven Eclips en Drools geen problemen. Drools voorziet een handleiding om de installatie vlot te laten verlopen, deze handleiding is alleen voor het besturingssysteem Windows voorzien maar veel grote verschillen zijn er niet met OS Mac.

Nadat beide applicaties werden afgehaald van de website staat er in de handleiding uitgelegd hoe men Drools moet installeren als plugin bij Eclipse. De instructies in de handleiding waren erg vaag en lieten dus nog wat denkwerk aan de gebruiker over. Na enige tijd experimenteren en het vragen van raad bij personen die reeds ervaring hadden met Eclipse, is de installatie van de plugin van Drools gelukt. Wellicht doen deze problemen zich in mindere mate voor bij geoefende en ervaren informatici. Men kan dus wel stellen dat de installatie van Drools en Eclipse enige informatica achtergrond eist.

Het openen van Eclipse met Drools als plugin geeft het beeld dat in figuur 4 getoond wordt.



Figuur 4 : Eclipse met Drools als plugin

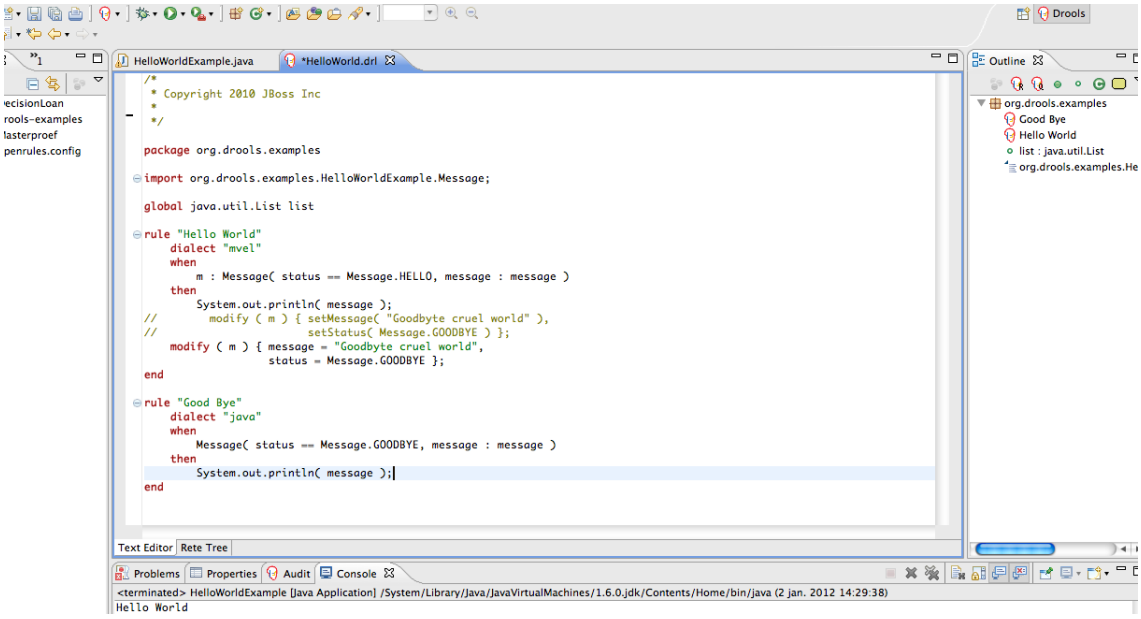
Drools voorziet enkele voorbeelden die men samen met de applicatie kan afhalen. Aan de hand van deze voorbeelden kan men makkelijker kennismaken met Drools zonder dat men van nul hoeft te starten met een eigen project.

a. Gebruiksvriendelijkheid van Drools bij een eerste kennismaking

i Zijn voorbeelden doeltreffend als kennismaking met Drools?

Voor iemand die niet helemaal thuis is in het programmeren, komt Drools meteen overweldigend over, zeker als men voorheen nooit kennis heeft gemaakt met Eclipse en Drools als plugin hier nog een boven op komt. Zoals voorheen gezegd kan men van start gaan met de voorbeelden. Deze maken het mogelijk om te experimenten met Drools en Eclipse. Eén van deze voorbeelden is het vaak ter illustratie gebruikte "Hello world" programma, dat goededag zegt tegen de persoon die het programma doet werken. Een weergave van dit voorbeeld wordt weergegeven in figuur 5, ook de rules voor dit programma worden hierin getoond. Later in dit hoofdstuk zal worden verduidelijkt wat men bedoelt met rules in Drools. Het is aangewezen de verschillende mogelijkheden die Eclipse en Drools te bieden hebben aan de hand van dit, of een andere voorbeeld, af te tasten. Bij deze voorbeelden moet men wel nog zelf de User Libraries toevoegen in Eclipse. Dit kan

vrij makkelijk door de instructies te volgen die op de Eclipse website worden aangeboden. (Pearson Education, 2004) De library is een reeks van .jar files die men aan een bepaald project moet toevoegen om het te kunnen uitvoeren, deze files zitten bij in het totale pakket dat men met de Drools software downloadt. Het toevoegen gaat zoals eerder gezegd erg eenvoudig. Het feit dat deze Libraries evenwel nog niet aanwezig zijn wordt echter niet vermeld in de handleiding van deze voorbeelden. Een ervaren informaticus zal de foutmelding die men krijgt wanneer de Libraries er niet zijn onmiddellijk herkennen en weten wat te doen. Gebruikers die hier voor de eerste keer kennis mee maken worden al onmiddellijk geconfronteerd met een probleem waarvan de oplossing niet meteen voor de hand ligt.



Figuur 5 : Het voorbeeld "Hello world" in Drools

ii Overzichtelijkheid van de Eclipse layout

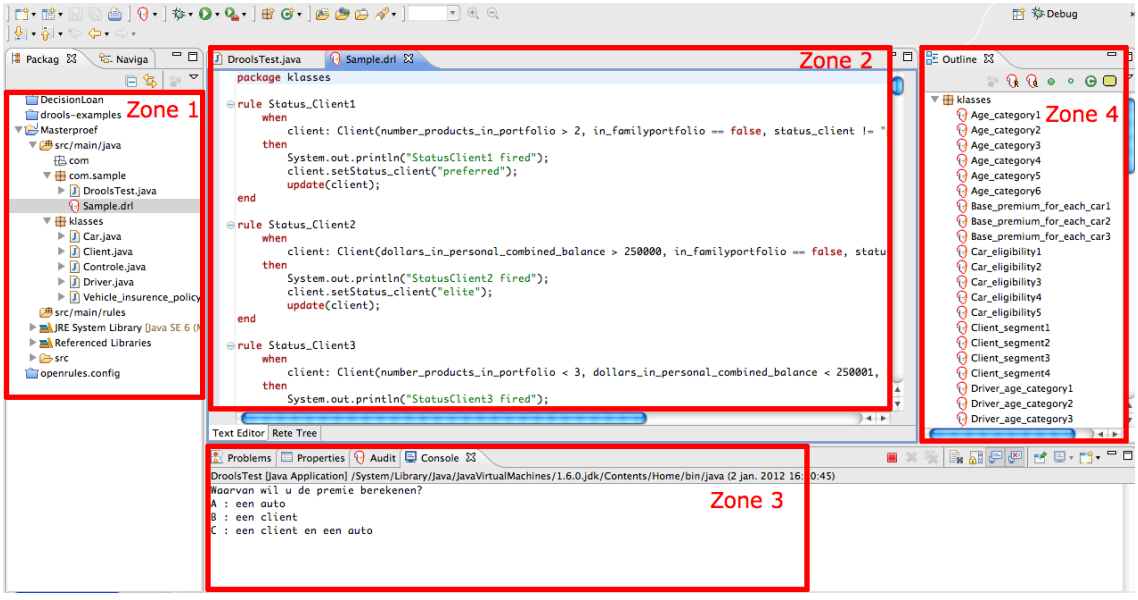
Eclipse biedt een zeer overzichtelijke layout zodat men een project volledig kan bekijken vanuit al zijn facetten. In figuur 6 wordt een overzicht gegeven van de visuele mogelijkheden die Eclipse met Drools als plugin te bieden heeft. In zone 1 van figuur 6 kan men steeds een overzicht terugvinden van al de projecten die men in Eclipse geïmporteerd heeft of die men heeft aangemaakt. Door op deze projecten te klikken kan men de verschillende onderdelen van deze projecten bekijken. Ook de libraries die hierboven vermeld werden treft men daar aan. Het project

“Masterproef” dat hier open staat is de implementatie van de Userv case waarover later meer.

De tweede zone in figuur 6 is deze waarin men op de verschillende tabbladen de code van een bepaald project kan bekijken. Op deze figuur staat het Drools tabblad open, hierin staat alleen de specifieke code die Drools toelaat. Om in deze code te kunnen navigeren is er een overzicht voorzien in zone 4. De Drools code wordt opgesplitst in verschillende Rules, in de Userv case zijn dit een tachtigtal Rules, het is dus erg moeilijk hier een overzicht van te krijgen of hierin te zoeken wanneer er aanpassingen moeten worden gedaan. Met de index die in zone 4 wordt weergegeven is het navigeren door de code heel wat eenvoudiger: men klikt de gewenste rule aan en vervolgens wordt deze rule weergegeven in zone 2.

De derde zone is deze waarin men onder andere de uitvoer te zien krijgt. Deze zone bestaat uit verschillende tabbladen waarin men ook foutmeldingen kan terugvinden.

Over het algemeen heeft Eclipse met Drools als plugin een zeer duidelijke layout, waarin de verschillende onderdelen apart worden aangeduid en het navigeren wordt vergemakkelijkt.



Figuur 6: Layout Eclipse met plugin Drools

C. Implementeerbaarheid van het Decision model in Drools .

In dit hoofdstuk wordt stilgestaan bij de implementatie van de Userv Case in Drools. Men wil hierbij achterhalen of Drools gebruiksvriendelijk is, welke voorkennis men nodig heeft om hiermee aan de slag te gaan en welke mogelijkheden Drools te bieden heeft. Verder zal worden gekeken of Drools extra mogelijkheden biedt die speciaal voor het Decision model werden ontwikkeld. Dit alles zal worden geverifieerd aan de hand van de Userv case. De verschillende struikelblokken en eveneens de mogelijke oplossingen hiervoor zullen worden besproken. Ten slotte worden ook de goede kanten van Drools uitvoerig besproken.

a. Vereiste voorkennis bij het werken met Drools

Zoals hierboven vermeld is Eclipse een Java ontwikkelingsomgeving, wat wil zeggen dat men alleen met de programmeertaal Java kan programmeren. Het is dus vereist dat men op de hoogte is van de Java syntaxis. Syntaxis is het geheel aan regels voor het vormen van toegestane combinaties van tekens en symbolen in een programmeertaal. (Slot Webcommerce bv.)

Het is dus belangrijk om weten dat bedrijven die met Drools willen werken, moeten beschikken over een informaticus die Java onder de knie heeft of die zich hierin wil bijscholen.

Extra voorkennis, boven op de Java kennis, is niet vereist om met Drools aan de slag te gaan.

b. Verloop van de implementatie van de Userv case in Drools.

De gebruiksvriendelijkheid van Drools nagaan kan alleen door effectief met Drools aan de slag te gaan. In deze verhandeling werd daarom gebruik gemaakt van de Userv Case om dit na te gaan. De Rule Families die eerder werden opgesteld, tracht men hier te implementeren in Drools, zodat deze bij geldige input automatisch zouden worden verwerkt.

Bij het volgen van de handleiding die Drools ter beschikking stelt, werd duidelijk dat de implementatie kan worden opgesplitst in drie grote stukken code.

- Het hoofdbestand, dat bestaat uit de Java code die de andere twee stukken code zal aanroepen. In dit hoofdbestand wordt ook gecodeerd hoe men de invoer van de data zal doen.
- Het volgende code blok is het definiëren van de verschillende klassen, dit zijn de feit types die gebruikt worden in de Rule Familes
- Het laatste deel code is deze waar de rijen van de Rule Families in gecodeerd worden.

Deze drie delen zullen hieronder meer in detail worden besproken. Vervolgens zal er verder worden ingegaan op de mogelijke problemen die zich kunnen voordoen bij de implementatie van de Userv case in Drools.

i Opstellen van het hoofdbestand

Het hoofdbestand bestaat uitsluitend uit Java code. Deze code zou ervoor moeten zorgen dat de feit type definities kunnen geladen worden en dat men de rules aanroept.

Een eerste stuk van dit hoofdbestand bestaat uit de KnowledgeBase en de KnowledgeBuilder. De KnowledgeBase is een collectie van de gecompileerde definities, rules en processen die werden gecompileerd door de KnowledgeBuilder. Om deze te creëren is er een KnowledgeBuilderFactory en een KnowledgeBaseFactory nodig.

In de ClassPathResource geeft men het pad van het rule bestand, dat hieronder verder wordt besproken, weer. Het type van dit bestand word aangegeven door DRL wat staat voor Drools Rule Language. In figuur 7 wordt de code die men hiervoor nodig heeft weergegeven. Deze code komt van de Userv case. Het bestand Sample.drl bevat de rules die zijn afgeleid van de Rule families. Dit deel van de code brengt geen moeilijkheden met zich mee vermist de code terug te vinden is in de Drools handleiding. De rest van de code is standaard code en vervolgens dient men dus enkel het pad nog aan te passen. (The JBoss Drools team)

```
private static KnowledgeBase readKnowledgeBase() throws Exception {
    KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
    String path = new String();
    path = "//Users//Tash//School//Hasselt//Masterproef//Drools//Masterproef//src//main//java//com//sample//Sample.drl";
    kbuilder.add(ResourceFactory.newFileResource(path), ResourceType.DRL);

    KnowledgeBuilderErrors errors = kbuilder.getErrors();
    if (errors.size() > 0) {
        for (KnowledgeBuilderError error: errors) {
            System.err.println(error);
        }
        throw new IllegalArgumentException("Could not parse knowledge.");
    }
    KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
    kbase.addKnowledgePackages(kbuilder.getKnowledgePackages());
    return kbase;
}
```

Figuur 7: Java code van het hoofdbestand

In dit hoofdbestand moet men de invoer van de data coderen. Dit doet men door middel van de Java syntaxis. Hierbij heeft de programmeur zelf de keuze hoe de invoer zal worden gedaan: hoe meer de programmeur de Java code onder de knie heeft, hoe meer mogelijkheden men heeft voor deze invoer. Voor de Userv case werd gekozen voor manuele invoer. Hierbij worden aan de gebruiker van het programma verschillende vragen gesteld zodat men alle nodige data kan ingeven. Een voorbeeld van deze code is terug te vinden in Figuur 8. In de Userv case werd ook een lus ingebouwd die ervoor zorgt dat men bij niet valabele invoer een boodschap krijgt met een verzoek om een geldige invoer te doen.

Het is een minpunt dat het deel over de invoer niet terug te vinden is in de gebruikershandleiding van Drools. Men moet zelf uitzoeken hoe men de invoer wil doen. Beter zou zijn dit vooraf te programmeren zodat men via een database of via Excel bestanden de data kan ingeven.

```
public class DroolsTest {

    public static final void main(String[] args) {
        try {


            KnowledgeBase kbase = readKnowledgeBase();
            System.out.println("Waarvan wil u de premie berekenen?");
            System.out.println("A : een auto");
            System.out.println("B : een client");
            System.out.println("C : een client en een auto");
            BufferedReader br = new BufferedReader(new InputStreamReader(System.in));
            String invoer = null;
            try {
               invoer = br.readLine();
            } catch (IOException ioe) {
            }
            while ( (invoer.contentEquals("A") == false) && (invoer.contentEquals("B")==false) && (invoer.contentEquals("C")==false )) {
                System.out.println( "Ongeldige invoer , voer A B of C in !");
                try {
                   invoer = br.readLine();
                } catch (IOException ioe) {
                }
            }
```

Figuur 8 : Data invoer

ii Definiëren van de feit types

Een tweede groot deel van de programmeer code is het deel waarin men de attributen moet definiëren. De attributen zijn de feit types die in bijlage 1 worden opgelijst. Deze attributen behoren steeds tot een bepaalde klasse. Zo behoren de attributen potential_occupant_injury_rating, on_high_theft_probability_list en make_and_model, samen met nog een aantal andere attributen, tot de klasse Car. Voor de Userv case zijn er vier verschillende klassen :

- Car
- Driver
- Client
- Vehicle_insurence_policy

Dit zijn de vier klassen waarover de feit types uit het Decision model verdeeld worden. Voor de Userv case werd er ook een vijfde klasse aangemaakt, namelijk Controle. Deze klasse dient slechts ter ondersteuning van de rules en zal verder worden verduidelijkt in het volgende deel.

Voor elk feit type dient te worden opgegeven tot welk data type het behoort. Men hoeft dus niet expliciet de mogelijke domein waarden op te geven, wat aan de programmeur extra vrijheid geeft en latere aanpassingen eenvoudiger maakt. Alleen in de rules hoeft men de feit type waarden weer te geven. Mogelijke data types zijn :

- String, dit is een opeenvolging van karakters
- Integer, dit is een geheel getal
- Char , is één karakter
- Boolean , kan de waarde “true” of “false” aannemen

(Hollander, 2008)

Bovenstaande types zijn de meest gebruikte datatypes. Java voorziet daarnaast ook nog een aantal andere types maar hiervan wordt in de Uvers case geen gebruik gemaakt. Hoe meer Java ervaring een programmeur heeft, hoe groter het pakket aan data types waarmee men kan werken zal zijn. In figuur 9 wordt getoond hoe men aan de attributen van de klasse Car een bepaald data type toewijst.

```
public class Car{
    public boolean convertible;
    public int price;
    public boolean on_high_theft_probability_list;
    public String potential_occupant_injury_rating;
    public String potential_theft_rating;
    public String type_of_airbag;
    public boolean roll_bar;
    public String car_eligibility;
    public boolean car_housed_same_state_owner;
    public String type_of_car;
    public String age_of_car;
    public boolean alarm;
    public boolean year;
    public String make_and_model;
```

Figuur 9 : Data types van de attributen in klasse Car

Als men aan al deze attributen een data type heeft toegewezen, moet men ook definiëren hoe deze attributen kunnen worden aangeroepen of hoe men er een waarde zal aan kunnen toewijzen. Indien gewenst doet Eclipse dit zelf, door slechts een enkele muisklik verschijnt de volledige code op het scherm. In figuur 10 wordt deze code weergegeven voor het attribuut Convertible. Bovenaan in figuur 10 staat ook gedefinieerd hoe men een instantie van de klasse Car moet aanmaken. Een instantie van Car wil zeggen dat men een object van de klasse Car aanmaakt met specifieke waarden voor de attributen die tot de klasse Car behoren. Zo kan men oneindig veel instanties van een bepaalde klasse aanmaken, die van mekaar kunnen worden onderscheiden door de waarden van hun attributen.

```
public Car(){
}
public void setConvertible(boolean convertible) {
    this.convertible = convertible;
}
public boolean isConvertible() {
    return convertible;
}
```

Figuur 10 : Java code van klasse Car en attribuut Convertible

Dit deel van de code is vrij eenvoudig te programmeren niettegenstaande er in de gebruikershandleiding van Drools geen aandacht aan wordt besteed.

iii Coderen van de Rules

Tot slot is er het belangrijkste deel code in Drools, als men het over het Decision model heeft. Dit is het stuk code waarin de logica van de Rule Families wordt weergegeven. In bijlage 4 worden alle rules voor de Userv case weergegeven.

Een belangrijk gegeven is hier dat de Java syntaxis niet volledig wordt gevolgd. De Rule Families worden in Drools weergegeven door middel van een “when, then” structuur. Dit is een structuur die Java niet kent, maar door de plugin van Drools

herkent Eclipse deze structuur wel. Voor elke rij in een Rule Family wordt er een rule aangemaakt. Één rule ziet er als volgt uit (figuur 11).

```
rule Potential_theft_category1
    when
        car: Car(convertible == true, potential_theft_rating !="high")
    then
        System.out.println("Potential_theft_category1 fired");
        car.setPotential_theft_rating("high");
        update(car);
end
```

Figuur 11 : Drools rule

Om te verduidelijken wat deze rule weergeeft, zal deze regel per regel overlopen worden.

- Elke rule start met het woord "rule", dit wordt herkend door Drools en verschijnt dan ook onmiddellijk in het rood. Vervolgens wordt de naam van de rule weergegeven. Elke rule moet een unieke naam hebben.
- Het volgende woord dat Drools in deze rule structuur herkent, is "when". Hierbij komen de voorwaarden waaraan voldaan moet zijn om tot een bepaalde conclusie te komen. Dit zijn dus de conclusie kolommen uit de Rule Familie. In dit voorbeeld moet de wagen convertible zijn en mag de potential_theft_rating nog niet op high staan. Deze laatste voorwaarde heeft niets met de Rule Familie te maken maar voorkomt oneindige lussen.
- Op de volgende regel vindt men de "then" terug die ook door Drools wordt herkent. Eerst en vooral staat hier een zin die wordt weergegeven als deze rule wordt uitgevoerd. Het is niet noodzakelijk deze zin weer te geven maar het maakt het achterhalen van fouten makkelijker. De volgende regel geeft de conclusie kolom van de Rule Family weer. Voor dit concreet voorbeeld wordt High als feit type waarde toegewezen aan het conclusie feit type potential_theft_rating. Tot slot wordt er in het "then" gedeelte ook een update gedaan. Die eveneens door Drools wordt herkend. Door deze update zal drools opnieuw alle rules gaan controleren waarin de klasse, die tussen haakjes wordt weergegeven, voorkomt. Deze update is ook de reden waarom in het vorig punt aangehaald werd dat lussen moeten worden vermeden. Deze update zou er anders voor zorgen dat alle rules, waarvan de condities voldaan zijn, oneindig aantal keren zouden worden uitgevoerd.
- Ten slotte wordt elke rule beëindigd met een "end"

Er werd eerder al aangehaald dat er nog een vijfde klasse Controle werd aangemaakt voor de Userv case. Het attribuut flag van deze klasse wordt gebruikt in de rules waarvan in de Rule Familie tot meerdere conclusies gekomen wordt bij één bepaalde invoer. Dit is het geval bij de Rule Family "premium increase for each car", die ook weergegeven staat in bijlage 2. De premie zal met een totaal bedrag verhoogd worden, dit bedrag is de som van al de conclusies uit deze Rule Family. Om te voorkomen dat er een oneindige lus ontstaat die deze rules oneindig veel keren blijft testen, wordt er een attribuut "flag" op een bepaalde waarde gezet na het uitvoeren van de rule. Bij de conditie van de rule wordt dan getest of dit attribuut flag reeds deze waarde heeft, zo niet wordt de rule uitgevoerd. Indien een attribuut deze waarde reeds heeft, wordt de rule niet meer uitgevoerd. Ter verduidelijking wordt het voorbeeld van een rij uit de Rule Family "premium increase for each car" getoond in figuur 12. In dit voorbeeld wordt aan het attribuut flag de waarde "premium_increase_for_each_car1" toegekend wanneer de rule wordt uitgevoerd. Wanneer bij de update dan opnieuw de condities voor deze rule worden getest, is niet aan alle condities voldaan, omdat flag reeds de waarde "premium_increase_for_each_car1" heeft.

```
rule premium_increase_for_each_car1
    when
        car: Car(age_of_car == "model year is current year or next year")
        vehicle_insurence_policy: Vehicle_insurence_policy(score: premium_increase_for_each_car)
        not Controle(flag == "premium_increase_for_each_car1")
    then
        System.out.println("premium_increase_for_each_car1 fired");
        vehicle_insurence_policy.setPremium_increase_for_each_car(score + 400 );
        Controle controle = new Controle();
        controle.setFlag("premium_increase_for_each_car1");
        update(vehicle_insurence_policy);
        insert(controle);
end
```

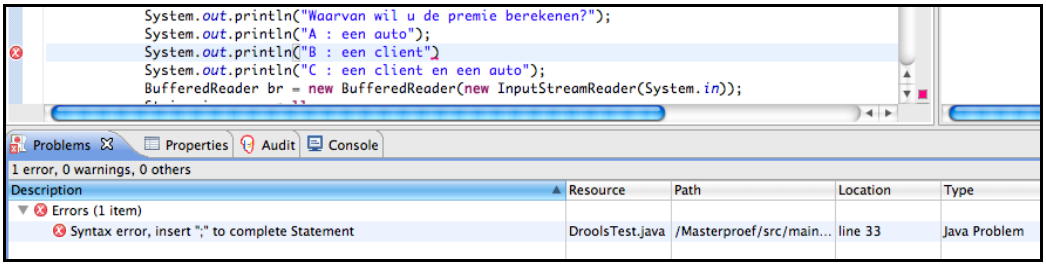
Figuur 12: Voorbeeld van het gebruik van het attribuut flag uit de klasse Controle.

iv Moeilijkheid van het oplossen van fouten in het programma

Indien men programma's aanmaakt met Java code of met andere programmeertalen, moeten deze programma's voldoen aan de syntaxis van deze specifieke programmeertaal. Bij het programmeren in Eclipse zijn er twee soorten fouten die zich kunnen voordoen.

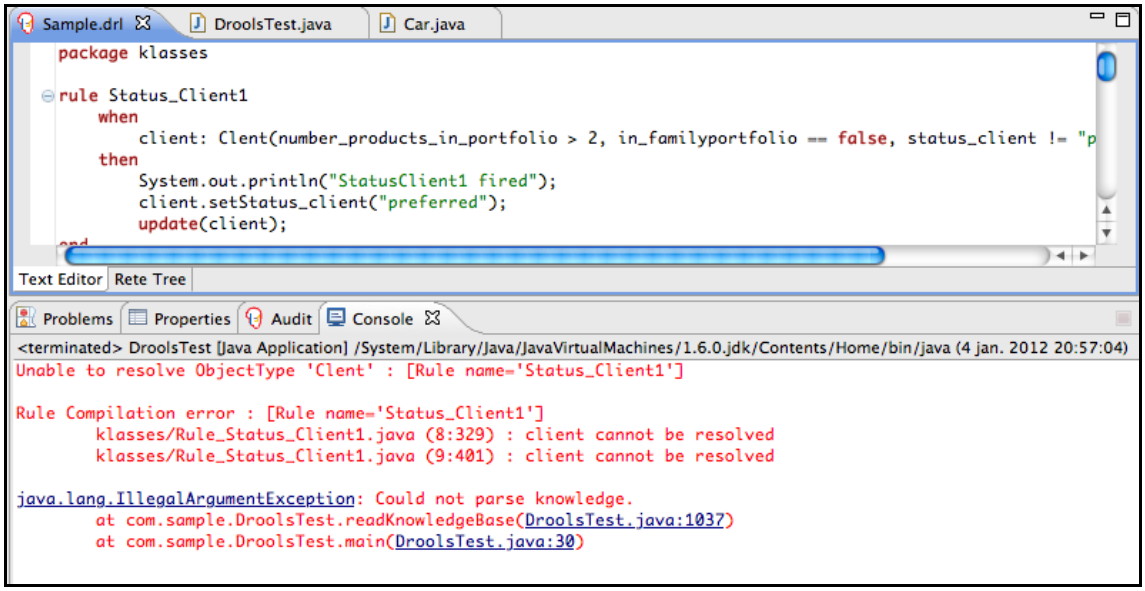
Een eerste soort wordt onmiddellijk aangegeven in de code met een rode stip, dit zijn de fouten in de Java code. Een voorbeeld hiervan is terug te vinden in figuur 13. Bovenaan in deze afbeelding wordt op de regel waar de fout zich voordoet een

aanduiding gegeven. Onderdaan bij het tabblad "Problems" wordt gerapporteerd welke fout er gemaakt wordt en waar in de code deze fout voorkomt.



Figuur 13 : Fout in Java code

Andere fouten worden pas bij het uitvoeren opgemerkt. Dit zijn de fouten die gemaakt zijn in de rule code, zoals getoond in figuur 14. Bij het uitvoeren komt er dan onderaan in het tabblad "Console" een error te staan. In deze error wordt weergegeven wat deze fout inhoudt en waar deze fout wordt gemaakt.



Figuur 14: Fout bij het uitvoeren in de rule code

Doordat Eclipse en Drools steeds aangeven waar een bepaalde fout wordt gemaakt en welke fout men maakt, wordt het programmeren van een Decision model iets eenvoudiger. Ook bij de Userv case was dit het geval. Verwacht werd dus dat het programmeren van de rules geen problemen zou geven. Bij het programmeren van de Java code kunnen moeilijkere fouten gemaakt worden die al wat minder makkelijk te achterhalen zijn. Om ook deze fouten te kunnen opsporen is een

Debug optie bij Eclipse waarmee de code kan doorlopen worden en men bepaalde stops in kan lassen waar men de waarden van de attributen wil kennen.

Men stelt vast dat deze manier van fouten weergeven en de Debug optie ervoor zorgen dat Drools gebruiksvriendelijk is voor programmeurs, eveneens voor personen die wat minder ervaring hebben met programmeren.

c. Voldoet Drools aan de vereisten van het Decision model?

Om na te gaan of Drools kan voldoen aan de vijftien principes die aan de basis liggen van het Decision model zullen we de Userv case die geïmplementeerd is in Drools, aftoetsen aan de principes.

i Principe 1 en 2

De twee dimensionale tabel wordt niet weergegeven in Drools maar wordt in zekere zin wel verwezenlijkt. In elke rule wordt namelijk een rij weergegeven van een Rule Family, wanneer men de rule van één bepaalde Rule Family samenneemt, heeft men ook alle kolommen. Visueel is dit inderdaad geen twee dimensionale structuur maar de rules vertegenwoordigen deze wel.

Er is visueel ook geen hoofding te zien in de Drools code maar de hoofding is zoals eerder vermeld een set van feit types, deze set van feit types wordt ook weergegeven in de rules. In Figuur 15 wordt de rule Driver_age_category1 weergegeven. Deze is gebaseerd op de Rule Family Driver age category die weergegeven wordt in bijlage 2. In deze Rule Family vormen 'Training certification' en 'Age category' een set van feit typen die de hoofding bepalen. Als we dan figuur 15 bekijken zien we dat deze feit types ook terug komen in het 'when' gedeelte van de rule.

```
rule Driver_age_category1
    when
        driver: Driver(age_category == "young", training_certification == false,
        drivers_age_category != "young driver")
    then
        System.out.println("Driver_age_category1 fired");
        driver.setDrivers_age_category("young driver");
        update(driver);
end
```

Figuur 15 : Voorbeeld rule in Drools

ii Principe 3 en 4

De atomaire logische uitdrukkingen worden in Drools in het "when" gedeelte van de code weergegeven. De verschillende attributen worden afgetoetst aan hun mogelijke waarden door middel van een operator. De operatoren in de Drools code zien er omwille van de syntaxis niet hetzelfde uit als in de Rule Families maar hebben wel dezelfde betekenis. Zo is bijvoorbeeld de operator "==" in Drools gelijk aan "Is" in de Rule Families.

Wat principe 4 betreft wordt zoals eerder vermeld elke rij van een Rule Family weergegeven in Drools door een afzonderlijke rule. Elke feit type van een Rule Family behoort tot het 'when' gedeelte, wat betreft de condities, het 'then' gedeelte, de conclusie.

iii Principe 5,6 en 7

Zoals principe 5 vooropstelt, heeft elke Rule Family slecht één conclusie kolom. In Drools wordt dit evenwel niet afgedwongen, er is namelijk de mogelijkheid om meerdere conclusies te hebben binnen één rule. Het is bijgevolg aan de programmeur om dit principe bewust toe te passen. Drools laat het wel toe om maar één conclusie te hebben in de 'then' van een rule.

Een eerste deel van principe 6 geeft aan dat aan alle condities van een bepaalde rij van een Rule Family moeten voldaan zijn. Het gaat hier namelijk om een AND operator tussen de verschillende condities. In Drools wordt er ook van de And operator uitgegaan binnen de 'when' structuur van de rules. Aan alle de condities in het 'when' gedeelte moeten voldaan zijn alvorens men zal over gaan tot de 'then'. Vervolgens handelt principe 6 over Rule Patterns, deze kunnen niet weergegeven worden in Drools.

In principe 7 wordt gesteld dat de conclusie cellen niet leeg mogen zijn. De conclusie kolom komt in Drools overeen met het 'then' gedeelte van de rules. Drools dwingt niet af dat hier een conclusie in wordt gecodeerd, dit 'then' gedeelte kan ook leeg zijn. Drools biedt dus wel de mogelijkheid om aan dit principe te voldoen maar het is aan de programmeur zelf om rekening te houden met dit principe.

iv Principe 8,9 en 10

Deze drie principes zorgen ervoor dat het Decision model declaratief is. De volgorde waarin de rules of de condities in het 'when' gedeelte van een rule geprogrammeerd worden spelen geen rol, deze volgorde heeft geen betekenis.

Drools legt ook geen specifieke volgorde op aan de relaties tussen de verschillende rules, Drools ondersteunt namelijk zowel forward als backward chaining.

v Principe 11 en 12

Principe 11 gaat geneste structuren tegen. Deze geneste structuren worden echter niet geëlimineerd door Drools. De programmeur moet er zelf voor zorgen dat deze niet voorkomen, aangezien Drools wel geneste structuren toelaat, Drools kan wel rules weergeven die aan principe 11 voldoen.

Zoals eerder vermeld herkent Drools geen Rule Patterns waardoor heel wat van de subprincipes van principe 12 niet kunnen worden getest binnen Drools. Wel kan een Rule Family die aan deze subprincipes voldoet, gecodeerd worden in Drools.

Het subprincipe dat afdwingt dat elke Rule Family minstens tot één conclusie moet leiden, wordt niet afgedwongen door Drools. Als de input aan geen enkele conditie voldoet, zal een conclusie zijn beginwaarde behouden. Met beginwaarde wordt bedoeld dat elk data type een initiële waarde heeft. Bij een string is dit ""(leeg) , bij een integer is dit 0 en bij een boolean is dit "false". Wel is het mogelijk om bij de input af te dwingen dat alleen bepaalde waarden aan een attribuut kunnen worden toegekend. Wanneer men deze waarden zo kiest dat er steeds minstens aan de condities van één conclusie is voldaan, zal ook aan dit subprincipe voldaan zijn.

Het feit dat een Rule Familie meer dan één conclusie kan hebben is op zich geen probleem voor Drools, ook dit kan men immers programmeren. Men moet vooraf echter bepalen welke Rule families meerdere conclusies kunnen hebben bij éénzelfde invoer. Hiervoor biedt Drools twee mogelijke oplossingen:

- Voor elke Rule Pattern van deze Rule Families een ander conclusie feit types naam kiezen(in Drools een ander attribuut). De reden hiervoor is dat een attribuut slechts 1 waarde kan aannemen.

- Een andere mogelijkheid is het aanmaken van een attribuut van het datatype array, op deze manier zal er een lijst van de verschillende conclusies van deze Rule familie aangemaakt worden.

Het laatste subprincipe van principe 12 kan ook gerealiseerd worden met Drools, de programmeur kiest namelijk zelf welke waarden de conclusie attributen zullen krijgen.

vi Principe 13,14 en 15

Een Decision model dat aan principe 13 en 14 voldoet kan in Drools gecodeerd worden maar Drools zal deze principes niet zelf afdwingen.

Principe 15 kan men meer zien als zijnde dat Drools moet voldoen aan wat het bedrijf ervan verwacht. Zal Drools ervoor zorgen dat het makkelijker word om bepaalde beslissingen te nemen? Worden deze beslissingen overzichtelijker? Voor de Userv case zou dit inderdaad het geval zijn, het Decision model zorgt er immers in de eerste plaats voor dat het geheel aan condities en conclusies overzichtelijk wordt. Wanneer dit Decision model dus geïmplementeerd wordt in Drools worden beslissingen automatisch weergegeven indien men een valabele input ingeeft.

vii Conclusie : Kan Drools al deze principes waarborgen?

Bij de implementatie van een Decision model in Drools kan in het algemeen gesteld worden dat Drools alle functionaliteiten bied die vereist zijn in de vijftien principes. Daarbij is het evenwel belangrijk aan te stippen dat Drools deze principes niet zelf afdwingt. Drools is niet specifiek toegepitst op het Decision model maar is voorzien om beslissingstabellen te automatiseren.

Bepaalde structuren die het Decision model kent, zoals een hoofding, een Rule Pattern, .. enz. worden bovendien niet visueel weergegeven in Drools.

d. Het resultaat van de implementatie van de Userv case in Drools

Men weet nu dat de implementatie van de Userv case in Drools net zoals het Decision model, aan alle principes zou moeten voldoen. Uiteraard is het ook belangrijk om het uiteindelijke resultaat van deze implementatie te bekijken, het is namelijk het resultaat hiervan waarmee een bedrijf effectief aan de slag zal gaan. Het resultaat zou ervoor moeten zorgen dat men aan efficiëntie en effectiviteit wint op het vlak van beslissingen nemen. Met efficiëntie wordt hier bedoeld dat men

doelmatig te werk gaat vanuit de idee dat het bedrijf ervoor wil zorgen dat het beslissingsproces zo vlot mogelijk verloopt. Men wil voorkomen dat er onnodige tijd en energie gestoken wordt in de manier waarop men beslissingen neemt. Met het verbeteren van de effectiviteit wil men bereiken dat er doeltreffend wordt gewerkt. Er moet bepaald worden wat men wil beslissen en deze beslissing moeten eenduidig en duidelijk zijn. (Slot Webcommerce bv.) Concrete wil dit voor Drools zeggen dat tot beslissingen moet worden gekomen met een zo klein mogelijke inspanning maar dat deze beslissingen wel zo correct mogelijk zouden moeten zijn.

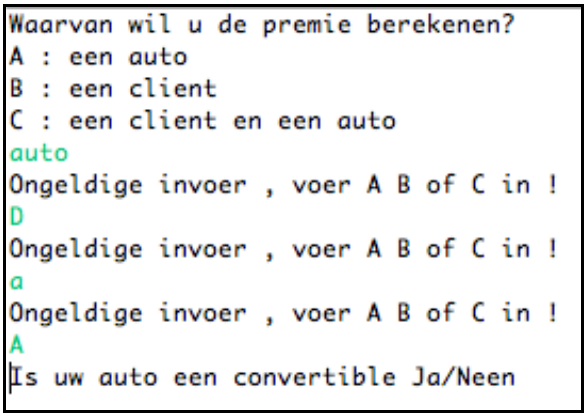
Als de volledige code van de implementatie geschreven is en er zitten geen fouten meer in dan kan men de code gaan testen aan de hand van test data. Dit is wat hier zal gedaan worden met de Userv case. Zoals eerder reeds werd vermeld, kunnen de input data van Excel bestanden, databases of andere bronnen komen. Concreet voor de Userv case worden manueel gegevens ingegeven die aan de gebruiker gevraagd worden. Dit wordt gedaan door middel van een keuze menu. Een stuk van deze input is te zien in figuur 16.

```
Waarvan wil u de premie berekenen?
A : een auto
B : een client
C : een client en een auto
A
Is uw auto een convertible Ja/Neen
Neen
Wat is de Prijs van uw auto? ($ niet toevoegen)
23000
Welke type airbag heeft u?
A : driver's airbag?
B : driver's and front passenger airbag
C : driver's front passenger and side panel airbag?
D : Geen
B
Heeft u een rollbar Ja/Neen
Neen
Is de auto in dezelfde staat ingeschreven als de eigenaar? Ja/Neen
Ja
Wat is het type van de auto
A : compact
B : sedan
C : luxury
D : andere
C
```

Figuur 16: Invoer van de gegevens.

Vermits de gegevens hier manueel worden ingevoerd, is het mogelijk dat er fouten worden gemaakt bij de invoer. Indien dit het geval is, zal er een foutboodschap worden gegeven en wordt aan de gebruiker gevraagd een nieuwe invoer te doen.

Een voorbeeld hiervan is te zien in figuur 17. Door dergelijke foutboodschappen kan men de efficiëntie verbeteren. Indien de foutboodschap er immers niet zou zijn, wordt een beslissing genomen op basis van foutieve gegevens en zal dus ook de effectiviteit verminderen.



Figuur 17 : Foutmelding bij foutieve invoer

Als voor elke vraag een geldige invoer werd gegeven zal het Drools programma met deze invoer een uitvoer bepalen. Deze uitvoer zal de uiteindelijke beslissing zijn. Voor de Userv case werden test data ingegeven waarvan in figuur 18 de uitvoer wordt getoond. De volledige invoer van de test data is terug te vinden in bijlage 5.

De eerste twaalf uitvoerregels geven een overzicht van welke rules er werden verwerkt. Dit zijn de rules waarbij aan alle condities en het "when" gedeelte voldaan was, waardoor ook het "then" gedeelte uitgevoerd werd. Er wordt hier niet weergegeven welke de conclusies van al deze rules waren, indien men dit wenst, is dit makkelijk bij te programmeren. In de Userv case wil men slechts enkele conclusies kennen, namelijk van deze die een uiteindelijk beslissing bevatten die in het Decision diagram in de achthoeken wordt geschreven. Deze beslissingen vindt men onderaan in figuur 18. Eerst en vooral wordt bepaald of iemand in aanmerking komt voor een verzekering of niet, vervolgens wordt bepaald hoe groot de premie zal zijn die deze persoon moet betalen.

Wanneer men de werking van Drools bekijkt op het niveau van het eindresultaat en niet de implementatie, kan men vaststellen dat de invoer voor de Userv case minder tijdrovend zou worden als deze niet manueel zou moeten worden ingegeven. Zoals eerder aangehaald kan dit evenwel nog worden gewijzigd na de

test fase. Eénmaal de invoer compleet is en men een eindbeslissing wil bekomen, kan men spreken van een zeer hoge mate van efficiëntie, er zijn namelijk geen extra handelingen meer vereist en het resultaat is binnen enkele seconden beschikbaar. Wanneer deze veronderstelling wordt veralgemeend naar andere cases toe, kan worden gesteld dat éénmaal de Drools code wordt gebruikt men een hoge mate van efficiëntie zou kunnen behalen. De invoer van data kan telkens op maat van het bedrijf gedaan worden, afhankelijk van hoe men deze data ter beschikking heeft. De verwerking van de invoer gebeurt dan snel, al zou dit voor een grotere hoveelheid dat of grotere Decision modellen meer tijd in beslag kunnen nemen. Drools zal de data evenwel sneller kunnen analyseren dan dat een persoon dit zou kunnen doen.

Wat betreft effectiviteit van het Drools programma, het programma zal bij correct invoer steeds als uitvoer een beslissing aanreiken. Deze beslissing zal eenduidig zijn en zeer duidelijk worden weergegeven. Voor de Userv case is er met de test data gecontroleerd of bij elke mogelijke combinaties van invoer gegevens ook de juiste beslissing als uitvoer wordt gegeven. Dit is een vereiste als men de effectiviteit wil verbeteren. Drools zou ervoor kunnen zorgen dat er steeds minder twijfel over bepaalde beslissingen zal zijn.

```
premium_discount_for_each_car1 fired
premium_increase_for_each_car2 fired
Base_premium_for_each_car3 fired
Client_segment1 fired
premium_increase_for_each_car5 fired
premium_increase_for_each_car4 fired
StatusClient1 fired
market_segment_discount1 fired
Car_eligibility5 fired
Vehicle_policy_eligibility_score3 fired
Potential_theft_category4 fired
Potential_occupant_injury_rating3 fired
-------------------------------------
-------------------------------------
De client is eligible for insurance
-------------------------------------
De basispremie voor de auto van deze client is $500
De basispremie wordt voor deze auto verhoogd met $1500
De basispremie wordt voor deze auto verlaagd met  15%
De basis premie wordt per bestuurder verhoogd met $0
De status van deze client is preferred
Op basis van deze status wordt de basispremie voor deze client verlaagd met $250
```

Figuur 18 : Output van de Userv case met test data

Hoofdstuk V : Implemenatie in OpenRules

A. Wat is OpenRules?

OpenRules is een Business Decision Management System(BDMS) en is een open source product. (OpenRules, Inc., 2003-2012) De voornaamste componenten van OpenRules zijn:

- Rule Repository : Enterprise-class Repository Maintained by Business Analysts with Excel® or Google Docs®
- Rule Learner : Discovering Rules, Predictive Analytics
- Rule Solver : Solving Constraint Satisfaction and Optimization Problems
- Rule Engine : Executing Decisions with Inter-Related Decision Tables
- Rule Dialog : Developing dynamic rules-based Web Questionnaires
- Finite State Machines : Event Processing, Connecting the Dots

Van OpenRules kan men alleen een test versie gratis afhalen van de website. Als men de volledige versie wil afhalen moet men 49,95$ betalen en als men hier ook de vernieuwingen voor een jaar bij wil moet men 99,95$ betalen. Vermits de testversie slechts beperkte mogelijkheden biedt, was het voor deze masterproef meer aangewezen de volledige versie van het programma te gebruiken. De beperkte versie voorziet bovendien geen mogelijkheid voor een plugin in Eclipse, heeft slecht een zeer beperkt aantal voorbeelden en voorziet geen ondersteuning.

Bij het aanschrijven van OpenRules, Inc. en het verduidelijken waarvoor deze volledige versie zou worden gebruikt in het kader van deze masterproef, werd na enige tijd de aanvraag tot een gratis volledige versie van OpenRules aanvaard. Hierbij werd een paswoord en een login aangereikt waarmee zowel de volledige versie van OpenRules als de nodige plugin voor Eclipse kon worden bemachtigd.

OpenRules, Inc. wil er vooral van uitgaan dat het automatiseren van beslissingen niet moet worden gedaan door programmeurs maar wel door bedrijfsanalisten. Hierbij wordt gebruik gemaakt van Excel voor het opstellen van zo goed als heel het programma.

B. Hoe verloopt de eerste kennismaking met OpenRules?

Het downloaden van OpenRules verloopt zeer vlot en wordt in enkele seconden beëindigd. Om van start te gaan met OpenRules voorziet men online enkele handleidingen die hulp kunnen bieden. Openrules heeft ook een specifieke handleiding voor het implementeren van het Decision model. In het algemeen kan het programma worden gebruikt voor het automatiseren van beslissingstabellen, evenals voor het Decision model. Bij het behandelen van de verschillende onderdelen waaruit een OpenRules programma bestaat wordt steeds teruggekoppeld naar het Decision model.

OpenRules werkt als plugin in Eclipse, maar deze plugin verandert niets aan hoe Eclipse er uitziet. De lay out blijft gelijkaardig aan hoe deze in het vorige hoofdstuk reeds werd beschreven. Het zorgt er alleen voor dat nu ook code in een Excel sheet kan worden geprogrammeerd.

a. Verloop van een eerste kennismaking met OpenRules

Zijn voorbeelden doeltreffend als kennismaking met OpenRules?
Bij de volledige versie van OpenRules worden heel wat verschillende voorbeelden aangereikt. Al deze voorbeelden karakteriseren een bepaalde eigenschap van het programma. Het is aangewezen alvorens men een eigen project start eerst eens met de voorbeelden de experimenteren. Deze voorbeelden kunnen ook een goede start zijn voor een eigen project, men kan er namelijk code uit overnemen of de structuur voor een Excel bestand uit afleiden.

Voor de gebruikers die Eclipse nog niet kennen, moet men een Eclipse handleiding raadplegen aangezien deze niet voorzien is bij OpenRules.

De Excel bestanden waarin het grootste deel van het programma geschreven worden zouden echter ook voor niet programmeurs vrij duidelijk moeten zijn. Een OpenRules programma oogt hierdoor onmiddellijk heel verstaanbaar voor heel wat mensen.

C. Implementeerbaarheid van het Decision model in Drools .

Dit deel zal handelen over de implementatie van de Userv Case in OpenRules. Concreet wil men achterhalen of OpenRules gebruiksvriendelijk is, welke voorkennis men nodig heeft om hiermee aan de slag te gaan en welke mogelijkheden OpenRules te bieden heeft. Daarnaast zal worden gekeken of OpenRules extra mogelijkheden biedt die speciaal voor het Decision model werden ontwikkeld. Dit alles zal opnieuw geverifieerd worden aan de hand van de Userv case. Net zoals bij Drools zullen de verschillende struikelblokken evenals de mogelijke oplossingen hiertoe worden besproken. Verder zullen ook de positieve kanten van OpenRules uitvoerig besproken worden. Voor ondersteuning aangaande de implementatie kan bij deze volledige versie gerekend worden op het support team van OpenRules.

a. Vereiste voorkennis bij het werken met OpenRules

Aangezien OpenRule een plugin is in Eclipse, is het belangrijk dat men een basis kennis heeft van de Java programmeertaal. OpenRules promoot zelf dat OpenRules door bedrijfsanalisten gebruikt zou worden en niet door programmeurs. Er is dus maar een zeer kleine Java kennis vereist. Uit de implementatie van de Userv case in OpenRules zal blijken hoe ver dezebasis kennis reikt.

Verder is het ook belangrijk dat men een basiskennis Excel heeft, men hoeft evenwel niet op de hoogte te zijn van alle functionaliteiten die Excel te bieden heeft. Kennis van het opstellen van tabellen met zelf ingevoerde tekst volstaat reeds.

b. Verloop van de implementatie van de Userv case in OpenRules.

Het nagaan van de gebruiksvriendelijkheid van OpenRules kan het beste gebeuren door er een case op toe te passen, dit werd gedaan met de Userv Case. De Rule Families die voorheen werden opgesteld worden nu geïmplementeerd in OpenRules, zodat deze bij geldige input automatisch worden verwerkt.

Bij het volgen van de handleiding die OpenRules ter beschikking stelt, werd duidelijk dat de implementatie kan worden opgesplitst in vijf grote stukken.

- Het hoofdbestand, dat bestaat uit de Java code die de Excel bestanden zal aanroepen.
- Het hoofdbeslissing Excel bestand waarin weergegeven wordt welke beslissingen er moeten worden genomen.
- Een Excel bestand met een overzicht van alle feit types en hun domein.
- De Rule Families worden ook in een apart Excel bestand opgesteld
- Tot slot wordt ook de invoer data in een Excel bestand meegegeven

Deze vijf delen zullen in wat volgt meer in detail worden besproken. Vervolgens zal er verder worden ingegaan op de mogelijke problemen die zich kunnen voordoen bij de implementatie.

i Het hoofdbestand

In dit hoofdbestand wordt gecodeerd hoe het hoofdbeslissingsExcel bestand moet aangeroepen worden. Er wordt weergeven waar dit bestand zich bevind en welke beslissing er bepaald moet worden. In dit bestand is het ook mogelijk om nog andere dingen in te programmeren, zoals de invoer of de uitvoer.

Voor de Userv case wordt er in dit hoofdbestand aan gegeven waar het Excel bestand "Decision" zich bevindt. (figuur 19) Er word een nieuwe beslissing aangemaakt die de naam " PremiumDecision" draagt en vervolgens wordt het commando gegeven om deze beslissing uit te voeren.

Main.java

```
package loan;

 * Developed by OpenRules Inc.

import com.openrules.ruleengine.Decision;

public class Main {
    public static void main(String[] args) {
        String fileName = "file:rules/main/Decision.xls";
        System.setProperty("OPENRULES_MODE", "Execute");
        Decision decision = new Decision("PremiumDecision",fileName);
        decision.execute();
    }
}
```

Figuur 19 : Het hoofdbestand in OpenRules

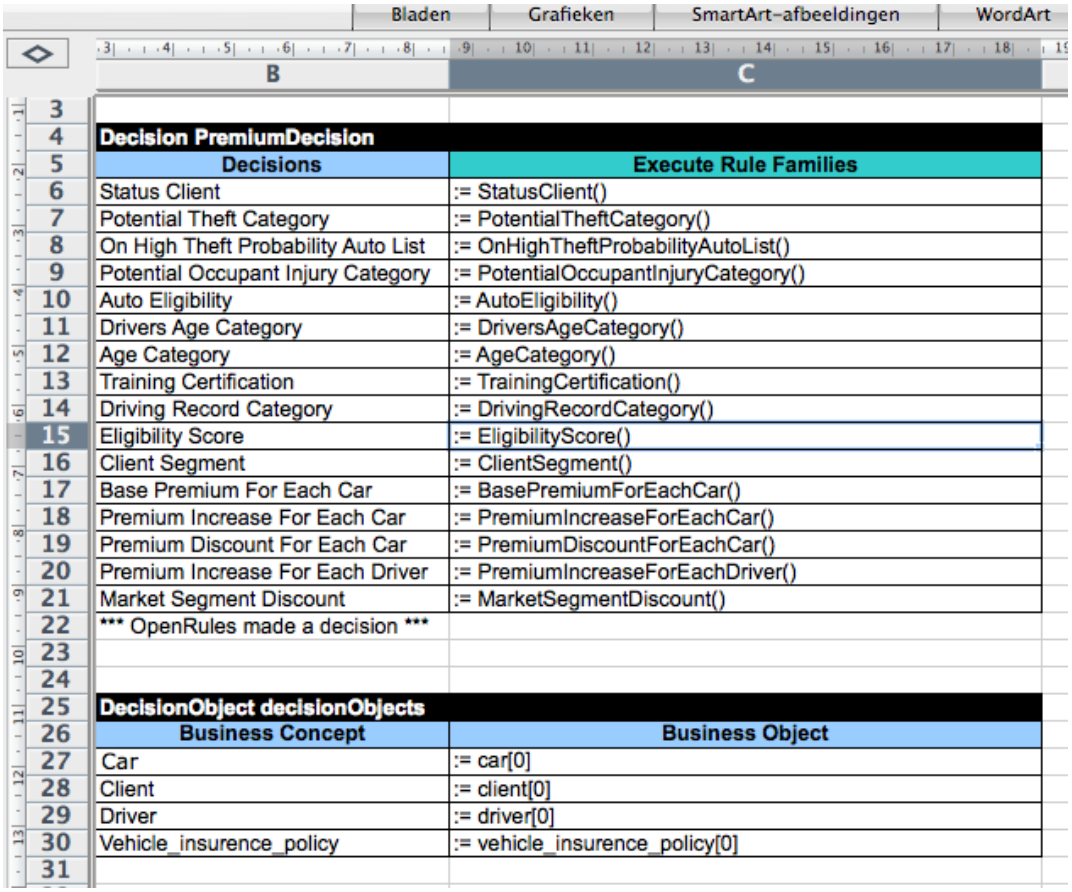
Dit is het enige bestand waarin men in de Java programmeer taal moet schijven. Het is een basis document dan men kan hergebruiken indien men de juiste namen wijzigt. Men zou ook nog extra's kunnen programmeren maar dit beslist men zelf, voor de basis applicatie heeft men alleen deze beperkte Java kennis nodig.

ii Het hoofdbeslissingsExcel bestand

In die Excel bestand worden alle beslissingen die moeten genomen worden opgesomd. Dit wil zeggen dat al de namen van de Rule Families worden weergegeven.

Voor de Userv case heet dit bestand PremiumDecision. Het bestand in getoond in Figuur 20. Er wordt eerst de naam van elke Rule Family gegeven en vervolgens hoe deze moet aangeroepen worden.

Dit bestand is makkelijk aan te maken door niet-programmeurs, men hoeft er niet voor te kunnen programmeren.



	B	C
3		
4	**Decision PremiumDecision**	
5	**Decisions**	**Execute Rule Families**
6	Status Client	:= StatusClient()
7	Potential Theft Category	:= PotentialTheftCategory()
8	On High Theft Probability Auto List	:= OnHighTheftProbabilityAutoList()
9	Potential Occupant Injury Category	:= PotentialOccupantInjuryCategory()
10	Auto Eligibility	:= AutoEligibility()
11	Drivers Age Category	:= DriversAgeCategory()
12	Age Category	:= AgeCategory()
13	Training Certification	:= TrainingCertification()
14	Driving Record Category	:= DrivingRecordCategory()
15	Eligibility Score	:= EligibilityScore()
16	Client Segment	:= ClientSegment()
17	Base Premium For Each Car	:= BasePremiumForEachCar()
18	Premium Increase For Each Car	:= PremiumIncreaseForEachCar()
19	Premium Discount For Each Car	:= PremiumDiscountForEachCar()
20	Premium Increase For Each Driver	:= PremiumIncreaseForEachDriver()
21	Market Segment Discount	:= MarketSegmentDiscount()
22	*** OpenRules made a decision ***	
23		
24		
25	**DecisionObject decisionObjects**	
26	**Business Concept**	**Business Object**
27	Car	:= car[0]
28	Client	:= client[0]
29	Driver	:= driver[0]
30	Vehicle_insurence_policy	:= vehicle_insurence_policy[0]
31		

Figuur 20 : Het Decision bestand van de Userv case

iii Overzicht feit types

Het derde bestand is een Excel bestand waarin alle feit types worden gecodeerd. Er wordt vermeld tot welk object elk feit type behoort en alle mogelijke waarden die een feit type kan aannemen worden weergegeven in de kolom "Domain". Er wordt duidelijk de link gelegd met de theorie van het Decision model, aangezien men ook de naamgeving van feit type heeft overgenomen.

Het op stellen van dit Excel bestand kan gedaan worden door iemand die de java programmeertaal niet machtig is.

In Figuur 21 wordt die Excel bestand voor de Userv case getoond. Zo is er bijvoorbeeld te zien in dit bestand at het feit type Convertible tot het object Car behoort, een instantie van dit feit type wordt dan weer zonder hoofdletter geschreven en ook de mogelijk waarden die converible kan aan nemen worden weergegeven, dit zijn Yes en No. Deze tabel wordt de Glossery genoemd, al kan men indien gewenst deze naam ook aanpassen.

Bladen | Grafieken | SmartArt-afbeeldingen

	A	B	C	D	E
1					
2		**Glossary glossary**			
3		**Fact Type**	**Object**	**Attribute**	**Domain**
4		**Convertible**		convertible	Yes,No
5		**Price**		price	0-1000000
6		**On_high_theft_probability_list**		on_high_theft_probability_list	Yes,No
7		**Potential_occupant_injury_rating**		potential_occupant_injury_rating	Extremely_high,High,Moderate,Low
8		**Potential_theft_rating**		potential_theft_rating	High,Moderate,Low
9		**Type_of_airbag**		type_of_airbag	Not_available,Driver_airbag,Driver_and_front_passenger_airbag,Driver_front_passenger_and_side_panel_airbag
10		**Roll_bar**		roll_bar	Yes,No
11		**Car_eligibility**	Car	car_eligibility	Not_eligible,Provisional,Eligible
12		**Car_housed_same_state_owner**		car_housed_same_state_owner	Yes,No
13		**Type_of_car**		type_of_car	Compact,Sedan,Luxury
14		**Age_of_car**		age_of_car	Model_year_is_current_year_or_next_year,Model_year_is_within_past_4_years,Model_years_is_between_past_4_years_and_past_10 years
15		**Alarm**		alarm	Yes,No
16		**Year**		year	1900-2015
17		**Make_and_model**		make_and_model	VWBug,Porsche,BMW,Other
18		**In_familyportfolio**		in_familyportfolio	Yes,No
19		**Status_client**		status_client	Preferred,Elite,Normal
20		**Number_products_in_portfolio**		number_products_in_portfolio	0-999
21		**Dollars_in_family_combined_balance**		dollars_in_family_combined_balance	0-1000000
22		**Dollars_in_personal_combined_balance**	Client	dollars_in_personal_combined_balance	0-1000000
23		**Products_in_familyportfolio**		products_in_familyportfolio	0-999
24		**Client_segment**		client_segment	Eligible_for_insurance,Reviewed_by_manager,Not_eligible_for_insurance
25		**Userv_portfolio_for_15_years**		userv_portfolio_for_15_years	Yes,No
26		**Age_category**		age_category	Young,Senior,Typical
27		**Training_certification**		training_certification	Yes,No
28		**Driving_record_category**		driving_record_category	High_risk_driver, No
29		**Accidents**		accidents	0-100
30		**Drivers_age_category**		drivers_age_category	Young_driver,Senior_driver,Eligible_driver
31		**Age**		age	16-115
32		**Sex**		sex	Male,Female

Figuur 21 : Glossery

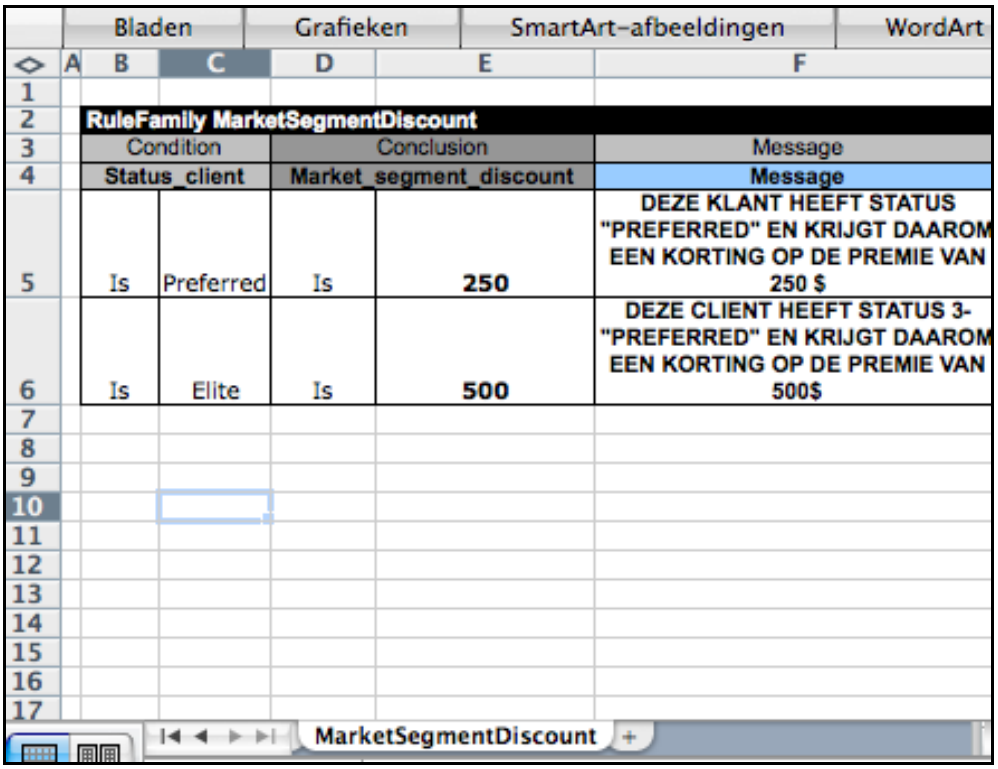
iv Het Rules bestand

Ook dit is een Excel bestand, voor dit onderzoek misschien wel het belangrijkste bestand. Hier staan namelijk de Rule Families in weergegeven. In Figuur 22 wordt de Rule Familie "MarketSegmentDiscount" van de Userv case getoond.

De opbouw van dit Excel bestand is steeds het zelfde.

- Slechts één Rule Family per tabblad, de naam van het tabblad moet ook de naam van de Rule Family zijn.
- De Rule Family start boven aan met een rij die de volledig Rule Family overspant en begint met de tekst " RuleFamily" , hiet kan eventueel ook "RuleFamilie1" of "RuleFamily2" staan, verder verduidelijking hierover volgt later. Dit woord wordt dan gevolgd door de naam van de Rule Family.
- Vervolgens worden de kolommen opgedeeld in condities, conclusie en/of bericht. De condties en de conclusie stemmen overeen met die van de Rule Families die reeds eerder werden opgesteld bij het volgen van de principes van het Decision model. De bericht kolom is niet verplicht, in deze kolom kan men een cijfer, woord of tekst schrijven die zal verschijnen wanneer aan de condities van deze conclusie voldaan is.
- Nu kan men de Rule Family verder aanvullen met de operatoren en de feit type waarden.

Het opstellen van dit Excel bestand met de Rule Families zou geen probleem mogen zijn voor bedrijfsanalisten net zoals OpenRules inc; verklaart. Excel is een gebruiksvriendelijke omgeving die niet veel extra kennis vereist.



RuleFamily MarketSegmentDiscount				
Condition		Conclusion		Message
Status_client		Market_segment_discount		Message
Is	Preferred	Is	250	DEZE KLANT HEEFT STATUS "PREFERRED" EN KRIJGT DAAROM EEN KORTING OP DE PREMIE VAN 250 $
Is	Elite	Is	500	DEZE CLIENT HEEFT STATUS 3-"PREFERRED" EN KRIJGT DAAROM EEN KORTING OP DE PREMIE VAN 500$

Figuur 22 : Rule Family "MarketSegmentDiscount" in OpenRules

v Data bestand

Tot Slot is er het data Excel bestand. Hierin kan de test data of de finale data weegegeven worden. Indien gewest kan men ook data uit andere bronnen halen , maar hiervoor is meer Java kennis vereist. Het hangt daarom af van de manier waarop men de data wil invoeren of er al dan niet een programmeur dit onderdeel moet aanmaken.

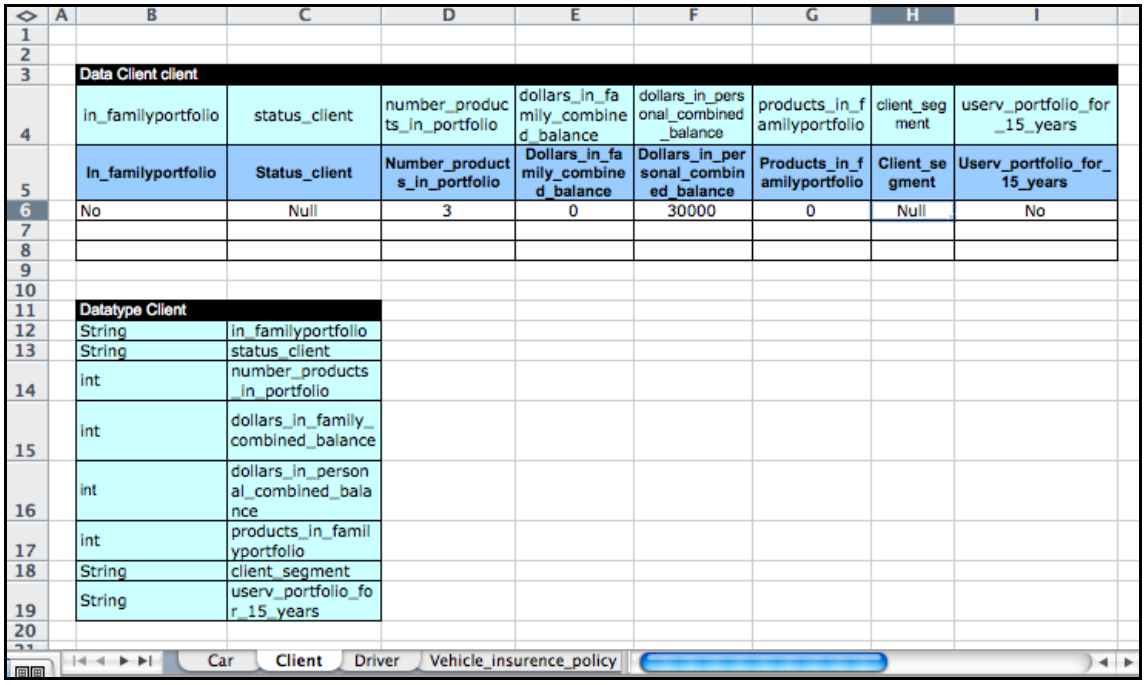
Voor de Userv case werd er zoals OpenRules aanraadt gekozen voor invoer via een Excel bestand. In figuur 23 wordt een deel van dit invoer bestand geïllustreerd, aangezien ook hier bepaalde vereisten gesteld worden zodat deze data kan worden ingelezen.

De data moet opgedeeld worden per object, elk object heeft een apart tabblad. In dit tabblad worden twee tabellen weergegeven:

- De data tabel, deze start met Data en de naam van het object waartoe de data behoort. Vervolgens wordt de naam van een instantie van een feit type gegeven en de feit type naam zelf. Deze tabel kan dan verder aangevuld worden met test data of finale data.

- De tweede tabel is deze waarin het data type van elke feit type instantie wordt vermeld. Hiervoor kan men gebruik maken van de data types ie Java aanreikt.

Voor het opstellen dat dit data Excel bestand heeft men een zeer beperkte Java kennis nodig , deze heeft men slecht nodig op data types toe te kennen, deze zouden makkelijk terug te vinden zijn in naslag werken over Java. In de Userv case worden maar drie data types gebruikt. Namelijk string, integer en boolean. (Hollander, 2008) In het deel over Drools werden deze reeds besproken.



	B	C	D	E	F	G	H	I
3	**Data Client client**							
4	in_familyportfolio	status_client	number_products_in_portfolio	dollars_in_family_combined_balance	dollars_in_personal_combined_balance	products_in_familyportfolio	client_segment	userv_portfolio_for_15_years
5	**In_familyportfolio**	**Status_client**	**Number_products_in_portfolio**	**Dollars_in_family_combined_balance**	**Dollars_in_personal_combined_balance**	**Products_in_familyportfolio**	**Client_segment**	**Userv_portfolio_for_15_years**
6	No	Null	3	0	30000	0	Null	No
7								
8								

	B	C
11	**Datatype Client**	
12	String	in_familyportfolio
13	String	status_client
14	int	number_products_in_portfolio
15	int	dollars_in_family_combined_balance
16	int	dollars_in_personal_combined_balance
17	int	products_in_familyportfolio
18	String	client_segment
19	String	userv_portfolio_for_15_years

Car | Client | Driver | Vehicle_insurence_policy

Figuur 23: Excel data invoerbestand voor OpenRules

vi Moeilijkheid van het oplossen van fouten in het programma.

Indien er zich fouten voordoen in de Java code dan worden deze fouten onmiddellijk aangegeven in het tabblad “Problem”, er wordt dan ook bij gemaild welke fout men maakt en waarde fout terug te vinden is.

De rest van het OpenRules programma bestaat niet uit Java code maar wel uit Excel bestanden, aangezien ook de fouten uit deze Excel bestanden moeten gehaald worden zou men eerst het programma moeten uitvoeren. Vervolgens zal OpenRules aangeven welke fout men maakt en waar deze fout zich voordoet. Om aan te geven waar deze fout zich voordoet worde de naam van het bestand

gegeven en de naam van het tabblad gevolgd door de cel waar de fout in gemaakt werd.

Bij wijze van voorbeeld is in Figuur 24 een fout melding te zien die gegeven werd bij het uitvoeren van het Userv OpenRules programma. Er wordt in deze foutmelding aangegeven dat er een fout zich in de Glossery , er wordt een feit type ingegeven in de andere Excel bestanden dat niet in de Glossery terug te vinden is.

Op deze manier worden de foutmelding eergegeven zodat ze door heel wat mensen zouden verstaan worden. Men hoeft geen programmeur te zijn om deze fout melding te snappen.

Figuur 24 : Foutmelding OpenRules

c. Voldoet OpenRules aan de vereisten van het Decision model?

Om na te gaan of OpenRules kan voldoen aan de vijftien principes die aan de basis liggen van het Decision model zullen we de Userv case die geïmplementeerd is in OpenRules toetsen aan de principes.

i Principe 1 en 2

De twee dimensionale tabel wordt ook in OpenRules weergegeven zoals deze in het eerste principe staat omschreven.

Er is visueel ook een hoofding te zien zoals principe twee eist.

ii Principe 3 en 4

Zoals principe 3 voorschrijft bevat elke cel een atomaire logische uitdrukking die conform is met de hoofding uit principe 2.

Wat principe 4 betreft wordt ook dit principe volledig gewaarborgd.

iii Principe 5,6 en 7

Net zoals principe 5 verlangt heeft elke Rule Family slecht één conclusie kolom. In OpenRules wordt dit niet afgedwongen, er is namelijk de mogelijkheid om meerdere conclusiekolommen te hebben binnen één Rule Familie. Het is dus aan de programmeur of bedrijfsanalist om dit principe bewust toe te passen.

Een eerste deel van principe 6 geeft aan dat alle condities van een bepaalde rij van een Rule Family moeten voldaan zijn, het gaat hier namelijk om een AND operator tussen de verschillende condities. In OpenRules wordt er ook van de AND operator uitgegaan, tussen de verschillende conditie kolommen. Vervolgens handelt principe 6 over Rule Patterns, deze kunnen niet weergegeven worden in OpenRules.

In principe 7 wordt gesteld wat de conclusie cellen niet leeg mogen zijn. OpenRules legt zelf wel de relaties tussen de kolommen van een zelfde feit type, maar het legt niet op dat een conclusie kolom niet leeg mag zijn. OpenRules biedt dus wel de mogelijkheid om aan dit principe te voldoen maar het is aan de programmeur of bedrijfsanalist zelf om rekening te houden met dit principe.

iv Principe 8,9 en 10

Deze drie principes zorgen ervoor dat het Decision model declaratief is. Wat Principe acht en toen betreft is er niet onmiddellijk een probleem, want hier wordt geen specifieke volorde opgelegd.

Maar Principe 9 is één van de grootste struikelblokken van OpenRules. Er zit namelijk wel steeds een volgorde in het lezen van de rijen van een Rule Families. Het is aan de persoon die de Rule Families opstelt om te bepalen hoe een Rule Family moet gelezen worden. OpenRules voorziet hier 3 soorten Rule Families.

- RuleFamily : De volledige rijen worden van boven naar beneden gelezen, als er één rij is waarvoor de condities allemaal voldaan zijn zal de conclusie van deze rij als conclusie voor de Rule Family genomen worden, de andere rijden zullen namelijk genegeerd worden.
- RuleFamily1: Alle rijen van een Rule Family zullen gelezen worden en de gene waarvan al de condities voldaan zijn worden op uitvoeren gezet. Als de volledige Rule Family gelezen is worden de conclusies van de uit te voeren rijen van boven naar beneden aan het conclusie feit type toegekend. Deze stoort Rule Family laat toe om de feit type waarden van een conditie te overschrijven. Maar indien deze

conclusie van de Rule Family voorkomt in een conditie van een andere Rule Family , zal deze afhankelijke Rule Family niet opnieuw uitgevoerd worden met deze nieuwe waarde.

- RuleFamily2: De rijen van de Rule Family worden van boven naar beneden gelezen, als er een rij tegen gekomen wordt waar de condities allen zijn voldaan wordt deze rij uitgevoerd. Integenstelling tot het vorige soort Rule Family kan een Rule Family die deze conclusie als conditie heeft wel opnieuw uitgevoerd.worden. Ook bij dit Rule Family type kunnen conclusie waarden overschreven worden.

Ongeacht welk soort Rule Family men kiest zal er steeds een volgorde zitten in het lezen van de rijen, waardoor aan principe 9 niet voldaan is.

v Principe 11 en 12

Principe 11 gaat geneste structuren tegen, deze geneste structuren worden niet aangegeven door OpenRules Men moet er zelf voor zorgen deze niet voorkomen aangezien OpenRules wel geneste structuren toelaat.

Zoals eerder vermeld herkent OpenRules geen Rule Patterns waardoor heel wat van de subprincipes van principe 12 niet kunnen getest worden binnen Drools. Wel kan een Rule Family die aan deze subprincipes voldoet, aangemaakt worden in OpenRules.

Het subprincipe dat afdwingt dat elke Rule Family minstens tot één conclusie moet leiden wordt niet afgedwongen door OpenRules. Wanneer de input aan geen enkele conditie voldoet zal een conclusie zijn begin waarde behouden zoals die in het data Excel bestand wordt weergegeven.

Het feit dat een Rule Family meer dan één conclusie kan hebben is op zich geen probleem voor OpenRules. Maar zoals hierboven reeds werd vermeld moet men zelf kiezen welk soort Rule Family men wil. Door van een conclusie feit type een rij van waarden te maken kan met meerder conclusies opnemen.

Het laatste subprincipe van principe 12 kan ook gerealiseerd worden met OpenRules, de programmeur kiest namelijk zelf welke waarden de conclusie attributen zullen krijgen.

vi Principe 13,14 en 15

Een Decision model dat aan principe 13 en 14 voldoet kan in Drools gecodeerd worden maar OpenRules zal deze principes niet zelf afdwingen.

Principe 15 kan men meer zien als zijnde dat OpenRules moet voldoen aan wat het bedrijf ervan verwacht. Zal OpenRules ervoor zorgen dat het makkelijker wordt om bepaalde beslissingen te nemen? Worden deze beslissingen overzichtelijker? Voor de Userv case zou dit inderdaad het geval zijn, het Decision model zorgt er in de eerste plaats voor dat het geheel aan condities en conclusies overzichtelijk wordt, als dit Decision model geïmplementeerd wordt in OpenRules worden beslissingen automatisch weergegeven indien men een valabele input ingeeft.

vii Conclusie : Kan Drools al deze principes waarborgen?

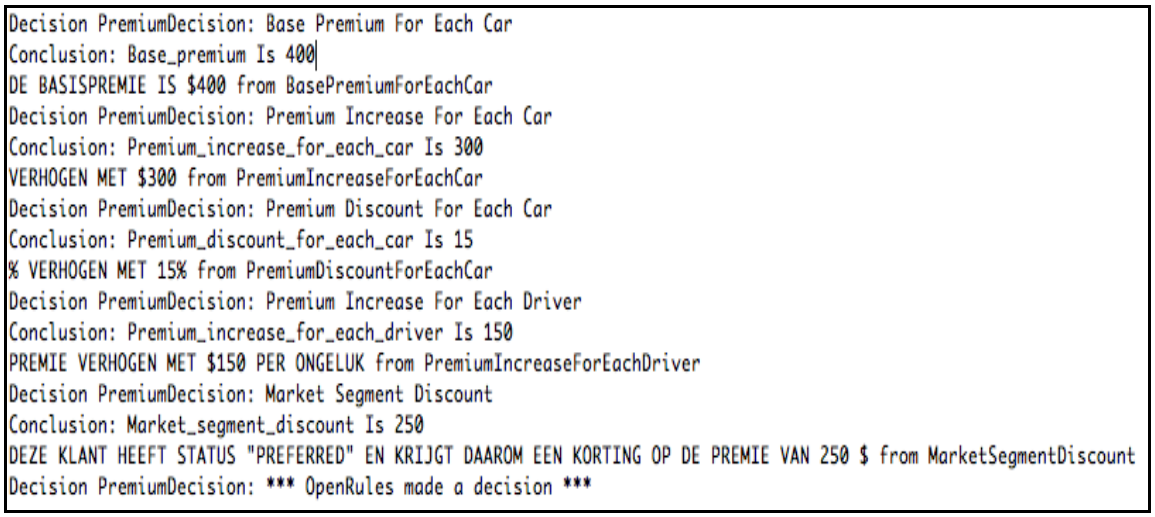
Over het algemeen voldoet OpenRules bijna aan alle principes. Alleen aan principe 9 is niet voldaan, maar met tracht dit te omzeilen door de verschillende soorten Rule Families. Visueel leunt OpenRules wel erg kort aan bij het Decision model.

d. Het resultaat van de implementatie van de Userv case in OpenRule.

OpenRules kan niet voldoen aan alle principes van het Decision model. Maar toch is het ook belangrijk om het resultaat van de implementatie te bekijken. Wat dit reslultaat zal door het bedrijf gebruikt worden om beslissingen te nemen. We zouden hier ook graag nagaan of OpenRules de efficientie en effectiviteit van beslissingen verbetert.

Als alle bestanden zoals voorheen vermeld worden opgesteld en de fouten er uitgehaald zijn kan men het programma uitvoeren. Men hoeft verder geen gegevens in te geven voor de Userv case, aangezien alle invoer data reeds in de invoer Excel staan.

Als het programma uitgevoerd wordt met de test data krijgt men de output die in figuur 25 wordt weergegeven. Hier is te zien dat elke keer wordt weergegeven welke Rule Family er wordt uitgevoerd. Vervolgens volgt dan een regel met de conclusie van deze Rule Family en op de regel daaronder worden de eventuele boodschappen weergegeven. Tot slot wordt in de laatste regel aangegeven dat er een eindbeslissing werd genomen en het programma wordt beëindigd. De volledige uitvoer wordt weergegeven in bijlage 6.



Figuur 25 : Uitvoer van OpenRules voor de Userv case

OpenRules verbetert de efficiëntie door het beslissingsproces te automatiseren, ook de effectiviteit wordt verbeterd aangezien er een duidelijke beslissing wordt aangereikt door het programma.

Hoofdstuk VI : Drools versus OpenRules

A. Het verschil en de gelijkenissen tussen de algemene eigenschappen

Drools en OpenRules hebben beide hetzelfde doel voor ogen, ze willen namelijk beide beslissingen automatiseren. In dit onderzoek werd gepoogd te achterhalen of ook de manier van beslissingen nemen, zoals gedefinieerd in het Decision Model, kan worden geautomatiseerd met Drools en OpenRules. Daarbij valt meteen een eerste verschil tussen beide programma's aan te stippen. Wanneer men kennis maakt met OpenRules en de introductie handleiding bekijkt of de site bekijkt, wordt er naar de theorie over het Decision model verwezen. OpenRules inc. besteedt duidelijk heel wat aandacht aan het op punt stellen van de software om op deze manier beslissingen te nemen. Drools daarentegen verwijst niet naar het Decision model.

Beide programma's zijn open source, wat echter geenszins wil zeggen dat ze beide gratis zijn. Van Drools kan men de volledige versie downloaden zonder hiervoor te moeten betalen, indien men hier ook ondersteuning bij wil moet er wel voor worden betaald. Het is evenwel ook mogelijk om vragen te stellen op het forum. Van OpenRules kan men slechts een beperkte evaluatie versie gratis verkrijgen. Voor de volledige versie en extra ondersteuning met vernieuwingen moet men betalen. Voor dit onderzoek werd evenwel gratis ondersteuning geboden via email en ook de wachttijden bij een vraag naar ondersteuning waren eerder beperkt.

Beide programma's ondersteunen zowel Windows als OS Mac.

B. Overeenkomsten en verschillen bij de implementatie

Bij het implementeren van het Decision model in Drools en OpenRules zijn er een heel aantal verschillen tussen beide programam's naar voren gekomen, al zijn er ook gelijkenissen aan te stippen.

a. Plugin in Eclipse

Beide programma's worden gebruikt als een plugin in de Java ontwikkelingsomgeving Eclipse. Dit wil zeggen dat er bij beiden een Java programmeertaal kennis nodig is. Toch is er een verschil; voor Drools heeft men heel wat meer Java kennis nodig dan voor OpenRules.

Het uitzicht van Eclipse is voor beide programma's is gelijkaardig, voor Drools wordt er apart tabblad voorzien waar in de rules worden geprogrammeerd. Voor OpenRules veranderd er daarentegen visueel niets aan Eclipse.

b. Data input en verwerking

De invoer van gegevens in Drools kan volledig door de gebruiker geschreven worden in Java, dit geeft de gebruiker uiteraard de vrijheid om zelf te kiezen hoe deze invoer zal gebeuren maar het vereist dan ook een uitgebreide Java kennis. Bij OpenRules reikt men de mogelijkheid aan om de data invoer via een Excel bestand te doen, waarvoor men geen Java nodig heeft. In dit opzicht is OpenRules dan ook meer gericht naar niet programmeurs terwijl Drools programmeurs vereist om het programma aan te maken.

Beide programma's beschikken wel over dezelfde mogelijkheden van data invoer. Het is echter veel complexer om de data invoer via Excel in Drools te doen dan in OpenRules.

Wat de verwerking van de data betreft, kan men op basis van de Userv case stellen dat dit bij beide programma's even snel gebeurt. Het is evenwel mogelijk dat de uitvoertijden zullen verschillen voor grotere cases, al kan dit niet met zekerheid worden bevestigd.

c. Rule Families

Eerst en vooral zijn er de principes waaraan deze Rule Families moeten voldoen volgens het Decision model. Drools kan voldoen aan bijna al de inhoudelijke principes, de principes worden niet afgedwongen door Drools maar ze zijn wel programmeerbaar. OpenRules daarentegen voldoet niet aan alle inhoudelijke principes, zo is principe 9 niet voldaan zoals het hoort; er moet namelijk steeds rekening worden gehouden met de volgorde waarin de rijen worden getest. Visueel

voldoet Drools dan weer niet aan alle principes, de condities en conclusies worden niet weergegeven in tabel vorm, wat bij OpenRules wel het geval is.

Voor het opstellen van de Rule Families in de programma's is bij beide programma's een duidelijke handleiding voorzien. Het is dan ook aan de gebruiker om te bepalen of hij zich meer thuis voelt in de Java taal, dan wel in Excel. Opnieuw moet hierbij worden opgemerkt dat de zelf te progammeren rules in Drools meer mogelijkheden bieden dan de vaste structuren in de Excel bestanden van OpenRules.

Biede programma's maken geen gebruik van Rule Patterns die in de theorie van het Decision model worden aangegeven.

d. Behandelen van foutmeldingen

Zowel Drools als OpenRules behandelen het geven van foutmeldingen op een zeer gebruiksvriendelijke manier. Er wordt bij alle twee de programma's aangegeven waar men de fout moet gaan zoeken, soms wordt er zelfs reeds een oplossing aangereikt.

Voor de persoon die het programma zal aanmaken is dit toch wel een belangrijke factor, aangezien er veel tijd zou verloren gaan bij het zoeken naar fouten indien deze niet op een duidelijke en begrijpbare manier zouden worden weergegeven.

e. Verschil in output

Bij Drools moet men zelf kiezen hoe men de output wil genereren, hier worden in de handleidingen geen instructies over gegeven. Het bepalen van deze uitvoer wordt volledig gecodeerd in de Java taal. Voor mensen met slecht een beperkte Java ervaring is deze manier van werken nadelig vermits men zelf op zoek moet gaan naar hoe men de output moet programmeren. Voordelig daarentegen, is dan weer dat men er op deze manier voor kan zorgen dat de output overzichtelijk is en wordt weergegeven in een formaat naar keuze.

Bij OpenRules werd reeds een manier van output voorzien in de basis handleiding. Indien gewenst, kan men zelf ook aanpassen hoe men de output wil genereren, al is het voor onervaren gebruikers meer aangewezen om de basis output te volgen. Hoewel deze weergavevorm niet heel overzichtelijk is, beschikt men wel over alle nodige gegevens.

Hoofdstruk VII :Conclusie

Het Decision model is een eenduidige en duidelijk begrijpbare theorie wat duidelijk bleek uit de toepassing van de theorie op de Userv case. Bij de toepassing van de theorie op deze case werden geen grote struikelblokken ontdekt, al bleek software voor het moddeleren van een Decision model diagram of Rule Families een tijdsbesparend instrument.

Zowel Drools als OpenRules zijn in staat om het Decision model te automatiseren. Hoewel elk programma zijn beperkingen heeft, moeten deze worden afgewogen tegen positieve eigenschappen van automatisering zoals tijdswinst, juistheid, ... enz. Indien men zich de vraag stelt of steeds één van beide programma's te verkiezen is boven het andere, is hiervoor geen sluitend antwoord te vinden. een keuze voor het ene dan wel het andere programma hangt af van de situatie waarin deze programma's zullen worden gebruikt. Drools vraagt een geruime Java voorkennis maar indien men over deze kennis beschikt, biedt het programma vele mogelijkheden. Wanneer men daarentegen de voorkeur geeft aan bedrijfsanalisten eerder dan aan programmeurs om met dergelijke programma's aan de slag te gaan, of wanneer men slechts een beperkte voorkennis van Java heeft, zal OpenRules een meer verdedigbare keuze zijn vermits Java in dit programma slechts beperkt aan bod komt. Verder is enkel kennis van Excel vereist om met OpenRules aan de slag te gaan. Wanneer men dan later alsnog wil uitbreiden is dit nog steeds mogelijk bij OpenRules indien men over de juiste Java kennis beschikt.

Besluitend kan men stellen dat een keuze voor een bepaald programma steeds zal worden ingegeven door de voorkennis van de persoon die met het programma aan de slag zal gaan. Beide programma's hebben hun sterktes en zwaktes en een keuze voor de sterkte van één programma houdt onvermijdelijk in dat men ook met de zwaktes van dat programma zal moeten rekening houden. Kiest men voor Drools, dan kiest men voor een inhoudelijk meer juiste oplossing maar schiet het programma visueel te kort. Kiest men voor OpenRules, dan kiest men voor een visueel sterke weergave, maar schiet het programma inhoudelijk dan weer deels te kort. 'Wie kiest verliest' zegt men dan.

Lijst van de geraadpleegde werken

Goldberg, B. v. (2010). The Decision Model #2 : Improving Proces Models and the Requirements proces.

Goldberg, B. v. (2010). The Decision Model , A Business Logic Framework Linking Business and Technology. Taylor and Francis Group.

Hollander, A. (2008). Java de basis. Pearson educations Benelux.

JBoss Enterprice. (z.j.). JBoss Community. Opgehaald van http://www.jboss.org/drools

Murk Schaafsma, W. v. (2011). Decision tables and rule engines in organ allocation systems for optimal transparancy ans flexibility. Tranplant international .

Newton, M., Cobb, J., Weaver, C., Kozmik, R., Krzyzanek, L., Chocholacek, J., et al. (z.j.). Opgehaald van Jboss Communty: http://www.jboss.org/drools

OpenRules, Inc. (2003-2012). Business Rules - Time to Excel. Opgehaald van OpenRules: http://openrules.com/

Pearson Education. (2004). Official Eclipse 3.0 FAQs. Opgehaald van Eclipse: http://wiki.eclipse.org/FAQ_How_do_I_add_an_extra_library_to_my_project%27s_classpath%3F

Slot Webcommerce bv. (z.j.). MWB. Opgehaald van http://www.mijnwoordenboek.nl

Snowden, D. J., & Boone, M. E. (2007, November). A leader's framework for decision making. Harverd Business Review .

Software, W. (2011). Opgeroepen op 10/6/2011, van http://www.wordwebonline.com

The Eclipse Foundation. (2011). Eclipse. Opgehaald van http://www.eclipse.org/

The JBoss Drools team. (2011). Drools expert User Guide. Opgehaald van http://docs.jboss.org/drools/release/5.4.0.Beta1/drools-expert-docs/html_single/index.html#d0e251

von Halle, B., & Goldberg, L. (2009). Evolving the decision Model with Views.

von Halle, B., & Goldberg, L. (2010). The decision model : A business logic framework linking business and technology. CRC Press Taylor & Francis Group.

von Halle, B., & Goldberg, L. (2009). The Decision Model 1: Linking Business leaders and technology.

Bijlage 1

Overzicht van de feit types en hun domein

Attribute	Domain
convertible	Yes,No
price	0-1000000
on_high_theft_probability_list	Yes,No
potential_occupant_injury_rating	Extremely_high,High,Moderate,Low
potential_theft_rating	High,Moderate,Low
type_of_airbag	Not_available,Driver_airbag,Driver_and_front_passenger_airbag,Driver_front_passenger_and_side_panel_airbag
roll_bar	Yes,No
car_eligibility	Not_eligible,Provisional,Eligible
car_housed_same_state_owner	Yes,No
type_of_car	Compact,Sedan,Luxury
age_of_car	Model_year_is_current_year_or_next_year,Model_year_is_within_past_4 _years,Model_years_is_between_past_4_years _and_past_10 years
alarm	Yes,No
year	1900-2015
make_and_model	VWBug,Porsche,BMW
in_familyportfolio	Yes,No
status_client	Preferred,Elite,Normal
number_products_in_portfolio	0-999
dollars_in_family_combined_balance	0-1000000
dollars_in_personal_combined_balance	0-1000000
products_in_familyportfolio	0-999
client_segment	Eligible_for_insurance,Reviewed_by_manager,Not_eligible_for_insurance
userv_portfolio_for_15_years	Yes,No
age_category	Young,Senior,Typical
training_certification	Yes,No
driving_record_category	High_risk_driver
accidents	0-100

drivers_age_category	Young_driver,Senior_driver,Eligible_driver
age	16-115
sex	Male,Female
marital_status	Married,Single
located_in_CA_NY_or_VA	Yes,No
kind_of_certification	From_school,Licensed_driver_training_company ,a_senior_citizen_driver's_refresher
history	Convicted_of_a _DUI,Number_of_accidents_is_greather_than_2 ,More_than_3 moving_voilations_in_the_last_2_years
eligibility_score	-1000-1000
base_premium	0-999
medical_coverage	Yes,No
uninsured_morerist_coverage	Yes,No
premium_increase_for_each_car	0-10000
premium_discount_for_each_car	0-100
premium_increase_for_each_driver	0-10000
market_segment_discount	0-2000

Status client

	Conditions										Conclusion	
Rule Pattern	Number of products in personal portfolio		Dollars in Personal protfolio Combined Balance		In family porfolio		Number of products in Family Portfoilio		Dollars in Family protfolio Combined Balance		Client Status	
1	Is greather than	2			Is	No					Is	Preferred
2			Is greather than	250000	Is	No					Is	Elite
3	Is Less Than	3	Is Less Than	250001	Is	No					Is	Normal
4					Is	Yes	>	2			Is	Preferred
5					Is	Yes			>	250000	Is	Elite
6					Is	Yes	<	3	<	250000	Is	Normal

Potential Theft Category

	Conditions						Conclusion	
Rule Pattern	Convertible car		Car's price		on " High Theft Probability Auto " list		Potential theft rating	
1	Is	Yes					Is	High
2			Is greather than	$ 45000			Is	High
3					Is	Yes	Is	High
4			Is between	($20000,$45000)	Is	No	Is	Moderate
4			Is less than	$20000	Is	No	Is	Low

on " High Theft Probability Auto " list

	Conditions				Conclusion	
Rule Pattern	Year		Make and model		on " High Theft Probability Auto " list	
1	Is between	(1964, 1970)	Is	VW Bug	Is	Yes
1	Is	any	Is	Porsche	Is	Yes
1	Is	any	Is	BMW	Is	Yes

Eligibility score

	Conditions								Conclusion	
Rule Pattern	Auto eligibility		Driver Age Category		Driving Record Category		Status Client		Vehicle Policy eligibility score	
1	Is	Not eligible							Increase by	100
1	Is	Provisional							Increase by	50
1	Is	Eligible							Increase by	0
2			Is	Young Driver					Increase by	30
2			Is	Eligible driver					Increase by	0
2			Is	Senior driver					Increase by	20
3					Is	High risk driver			Increase by	100
4							Is	Preferred	Decrease by	50
4							Is	Elite	Decrease by	100
4							Is	Normal	Decrease by	0

Client Segment

	Conditions				Conclusion	
Rule Pattern	Vehicle Policy eligibility score		Has Userv portfolio for 15 years		Client segment	
1	Is Less Than	100			Is	Eligible for insurance
1	Is between	(100, 251)			Is	Reviewed by manager
1	Is greather than	250			Is	Not eligible for insurance
2	Is	any	Is	Yes	Is	Eligible for insurance

Base Premium for each car

	Conditions		Conclusion	
Rule Pattern	Type of Car		Base Premium for each car	
1	Is	Compact	Is	$250
1	Is	Sedan	Is	$400
1	Is	Luxury	Is	$500

Premium increase for each driver

	Conditions										Conclusion	
Rule Pattern	Age category		Marital status		Located in CA,NY or VA		Driving Record Category		Number of accidents		Premium increase for each driver	
1	Is	Young	Is	Married	Is	Yes					Is	$700
1	Is	Young	Is	Single	Is	Yes					Is	$720
1	Is	Young	Is	Married	Is	No					Is	$300
1	Is	Young	Is	Single	Is	No					Is	$300
2	Is	Senior			Is	Yes					Is	$500
2	Is	Senior			Is	No					Is	$200
3	Is	Typical									Is	$0
4							Is	High risk driver			Is	$1000
5									Is greather than	0	Is	$150 * Number acc

Market segment discount

	Conditions		Conclusion	
Rule Pattern	Client Status		Market segment discount	
1	Is	Preferred	Is	$250
1	is	Elite	Is	$500

Premium increase for each car

	Conditions												Conclusion	
Rule Pattern	Age of car		Uninsured motorist coverage		Medical coverage		Car's potential injury rating		Potential theft rating		Car housed in same state as policy owner lives		Premium increase for each car	
1	Is	Model year is current year or next year											Is	$400
1	Is	Model year is within past 4 years											Is	$300
1	is	Model years is between past 4 years and past 10 years											Is	$250
2			Is	Included									Is	$600
3					Is	Included							Is	$600
4							Is	Extremely High					Is	$1000
4							Is	High					Is	$500
5									Is	High			Is	$500
6											Is	No	Is	$1000

Premium discount for each car

	Conditions						Conclusion	
Rule Pattern	Type of airbag		Potential theft rating		Equipped with alarm system		Premium discount for each car	
1	Is	Driver's airbag					Is	12%
1	Is	Driver's and front passenger airbag					Is	15%
1	Is	Driver's, front passenger and side panel airbag					Is	18%
2			Is	High	Is	Yes	Is	10%

Age Category

	Conditions				Conclusion	
Rule Pattern	Sex		Age		Age category	
1	Is	Male	Is less than	25	Is	Young
1	Is	Female	Is less than	20	Is	Young
1	Is	Male	Is greather than	70	Is	Senior
1	Is	Female	Is greather than	70	Is	Senior
1	Is	Male	Is Between	(25, 70)	Is	Typical
1	Is	Female	Is greather than	(20, 70)	Is	Typical

Training certification

	Conditions		Conclusion	
Rule Pattern	Kind of training		Training Certification	
1	Is	From school	Is	Yes
1	Is	Licensed driver training company	Is	Yes
1	Is	a senior citizen driver's refresher	Is	Yes
1	Is not in	From school, Licensed driver training company, a senior citizen driver's refresher	Is	No

Driving Record Category

	Conditions		Conclusion	
Rule Pattern	History		Driving Record Category	
1	Is	Convicted of a DUI	Is	High risk driver
1	Is	Number of accidents is greather than 2	Is	High risk driver
1	Is	More than 3 moving voilations in the last 2 years	Is	High risk driver

Potential Occupant Injury Category

	Conditions						Conclusion	
Rule Pattern	Type of airbag		Convertible car		Car has roll bar		Car's potential injury rating	
1	Is	Not available					Is	Extremely High
1	Is	Driver's airbag					Is	High
1	Is	Driver's and front passenger airbag					Is	Moderate
1	Is	Driver's, front passenger and side panel airbag					Is	Low
2			Is	Yes	Is	No	Is	High

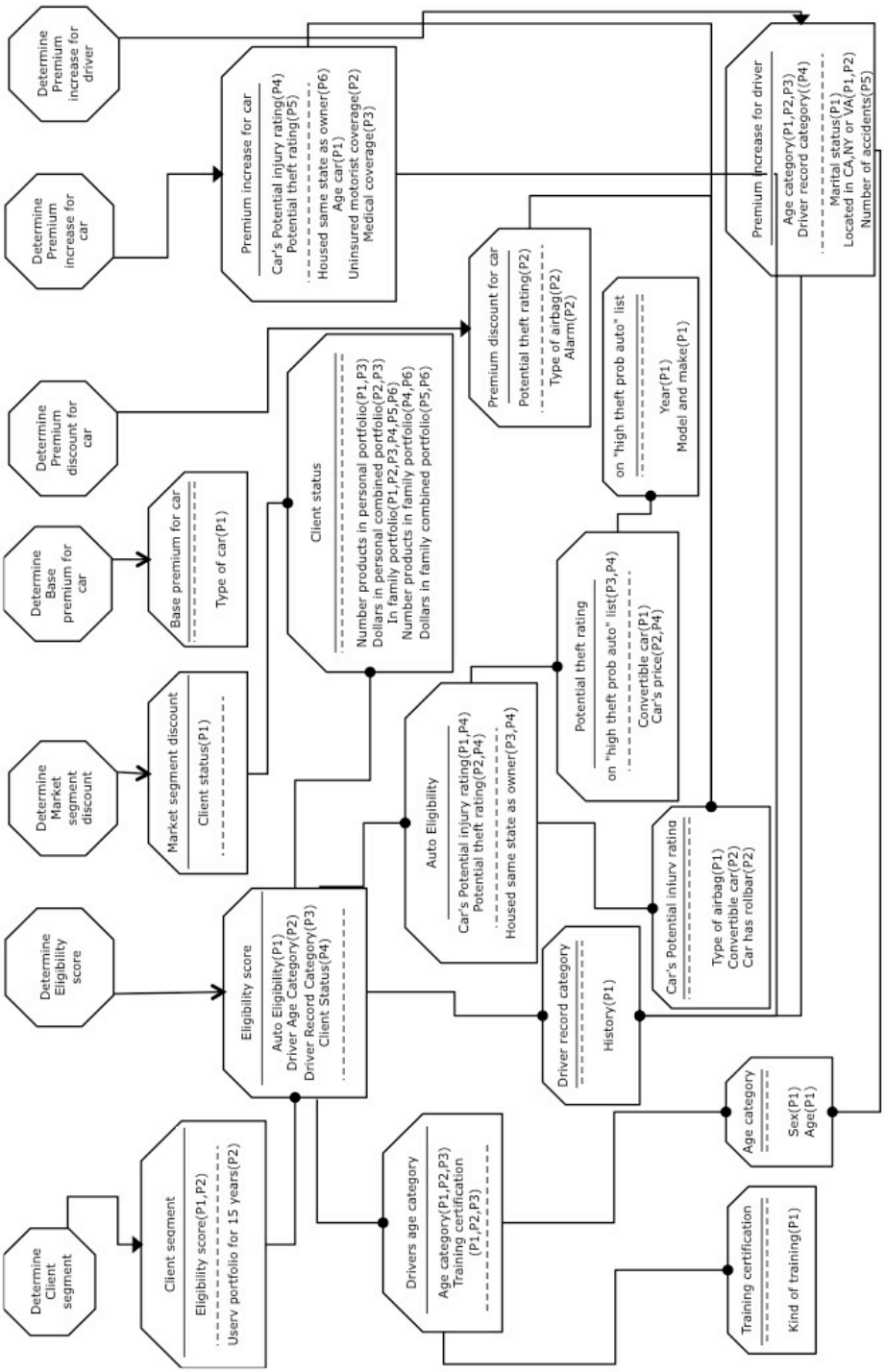
Auto Eligibility

	Conditions						Conclusion	
Rule Pattern	Car's potential injury rating		Potential theft rating		Car housed in same state as owner		Auto eligibility	
1	Is	High					Is	Not eligible
1	Is	Extremely High					Is	Provisional
2			Is	High			Is	Provisional
3					Is	No	Is	Provisional
4	Is not in	(High , Extremely High)	Is not	High	Is	Yes	Is	Eligible

Drivers Age Category

	Conditions				Conclusion	
Rule Pattern	Age Category		Training Certification		Driver Age Category	
1	Is	Young	Is	No	Is	Young Driver
1	Is	Young	Is	Yes	Is	Eligible driver
2	Is	Senior	Is	No	Is	Senior driver
2	Is	Senior	Is	Yes	Is	Eligible driver
3	Is	Typical	Is in	(Yes, No)	Is	Eligible driver

Bijlage 3


Determine Client segment
Client segment
Eligibility score(P1,P2)
Userv portfolio for 15 years(P2)
Determine Eligibility score
Eligibility score
Auto Eligibility(P1)
Driver Age Category(P2)
Driver Record Category(P3)
Client Status(P4)
Determine Market segment discount
Market segment discount
Client status(P1)
Determine Base premium for car
Base premium for car
Type of car(P1)
Determine Premium discount for car
Client status
Number products in personal portfolio(P1,P3)
Dollars in personal combined portfolio(P2,P3)
In family portfolio(P1,P2,P3,P4,P5,P6)
Number products in family portfolio(P4,P6)
Dollars in family combined portfolio(P5,P6)
Premium discount for car
Potential theft rating(P2)
Type of airbag(P2)
Alarm(P2)
Determine Premium increase for car
Determine Premium increase for driver
Premium increase for car
Car's Potential injury rating(P4)
Potential theft rating(P5)
Housed same state as owner(P6)
Age car(P1)
Uninsured motorist coverage(P2)
Medical coverage(P3)
Premium increase for driver
Age category(P1,P2,P3)
Driver record category((P4)
Marital status(P1)
Located in CA,NY or VA(P1,P2)
Number of accidents(P5)
Determine Market segment discount
Auto Eligibility
Car's Potential injury rating(P1,P4)
Potential theft rating(P2,P4)
Housed same state as owner(P3,P4)
Potential theft rating
on "high theft prob auto" list(P3,P4)
Convertible car(P1)
Car's price(P2,P4)
on "high theft prob auto" list
Year(P1)
Model and make(P1)
Car's Potential injury ratino
Type of airbag(P1)
Convertible car(P2)
Car has rollbar(P2)
Driver record category
History(P1)
Drivers age category
Age category(P1,P2,P3)
Training certification (P1,P2,P3)
Age category
Sex(P1)
Age(P1)
Training certification
Kind of training(P1)

Bijlage 4

```
import pack.Client;
import pack.Car;
import pack.Person;
import pack.Driver;
import pack.Vehicle_insurence_policy;


rule Status_Client1
        when
        client: Client(number_products_in_portfolio > 2, in_familyportfolio == false,
        status_Client != “preferred”)
        Then
        System.out.println("StatusClient1 fired");
        client.setstatus_Client(“preferred”);
        update(Client );

rule Status_Client2
        when
        client: Client(dollars_in_personal_combined_balance > 250000,
        in_familyportfolio == false, status_Client != “elite”)
        Then
        System.out.println("StatusClient2 fired");
        client.setstatus_Client(“elite”);
        update(Client );

rule Status_Client3
        when
        client: Client(number_products_in_portfolio < 3,
        dollars_in_personal_combined_balance < 250001, in_familyportfolio ==
        false, status_Client != “normal”)
        Then
        System.out.println("StatusClient3 fired");
        client.setstatus_Client(“normal”);
        update(Client );

rule Status_Client4
        when
        client: Client(in_familyportfolio == true, products_in_familyportfolio > 2,
        status_Client != “preferred”)
        Then
        System.out.println("StatusClient4 fired");
        client.setstatus_Client(“preferred”);
        update(Client );
```

```
rule Status_Client5
        when
        client: Client(in_familyportfolio == true,dollars_in_family_combined_balance
        > 250000, status_Client != "elite")
        Then
        System.out.println("StatusClient5 fired");
        client.setstatus_Client("elite");
        update(Client );

rule Status_Client6
        when
        client: Client(in_familyportfolio == true,dollars_in_family_combined_balance
        < 250001, products_in_familyportfolio < 3, status_Client != "normal")
        Then
        System.out.println("StatusClient6 fired");
        client.setstatus_Client("normal");
        update(Client );

rule Potential_theft_category1
        when
        car: Car(convertible == true, potential_theft_rating !="high")
        Then
        System.out.println("Potential_theft_category1 fired");
        car.setpotential_theft_rating ("high");
        update(Car);

rule Potential_theft_category2
        when
        car: Car(price > 45000, potential_theft_rating !="high")
        Then
        System.out.println("Potential_theft_category2 fired");
        car.setpotential_theft_rating ("high");
        update(Car);

rule Potential_theft_category3
        when
        car: Car(on_high_theft_probability_list == true, potential_theft_rating
        !="high")
        Then
        System.out.println("Potential_theft_category3 fired");
        car.setpotential_theft_rating ("high");
        update(Car);

rule Potential_theft_category4
        when
        car: Car(price >20000, price < 45001, on_high_theft_probability_list ==
        false, potential_theft_rating !="moderate")
        Then
        System.out.println("Potential_theft_category4 fired");
        car.setpotential_theft_rating ("moderate");
```

```
        update(Car);

rule Potential_theft_category5
        when
        car: Car(price < 20001, on_high_theft_probability_list == false,
        potential_theft_rating !="low")
        Then
        System.out.println("Potential_theft_category5 fired");
        car.setpotential_theft_rating ("low");
        update(Car);

rule On_high_theft_probability_list1
        when
        car: Car(year > 1964, year < 1971, make_and_model == "VW Bug",
        on_high_theft_probability_list != true)
        Then
        System.out.println("on_high_theft_probability_list1 fired");
        car.seton_high_theft_probability_list (true);
        update(Car);

rule 0n_high_theft_probability_list2
        when
        car: Car(make_and_model == "Porsche", on_high_theft_probability_list !=
        true )
        Then
        System.out.println("on_high_theft_probability_list2 fired");
        car.seton_high_theft_probability_list (true);
        update(Car);

rule 0n_high_theft_probability_list3
        when
        car: Car(make_and_model == "BMW", on_high_theft_probability_list !=
        true)
        Then
        System.out.println("on_high_theft_probability_list3 fired");
        car.seton_high_theft_probability_list (true);
        update(Car);

rule Potential_occupant_injury_rating1
        when
        car: Car(type_of_airbag == "not available",
        potential_occupant_injury_rating != "extremely high")
        Then
        System.out.println("Potential_occupant_injury_rating1 fired");
        car.setpotential_occupant_injury_rating ("extremely high");
        update(Car);


rule Potential_occupant_injury_rating2
        when
        car: Car(type_of_airbag == "driver's airbag",
```

```
        potential_occupant_injury_rating != “high”)
        Then
        System.out.println("Potential_occupant_injury_rating2 fired");
        car.setpotential_occupant_injury_rating (“high”);
        update(Car);

rule Potential_occupant_injury_rating3
        when
        car: Car(type_of_airbag == “driver’s and front passenger airbag”,
        potential_occupant_injury_rating != “moderate”)
        Then
        System.out.println("Potential_occupant_injury_rating3 fired");
        car.setpotential_occupant_injury_rating (“moderate”);
        update(Car);

rule Potential_occupant_injury_rating4
        when
        car: Car(type_of_airbag == “driver’s front passenger and side panel airbag”,
        potential_occupant_injury_rating != “low”)
        Then
        System.out.println("Potential_occupant_injury_rating4 fired");
        car.setpotential_occupant_injury_rating (“low”);
        update(Car);

rule Potential_occupant_injury_rating5
        when
        car: Car(convertible == true, Roll_bar == false,
        potential_occupant_injury_rating != “high”)
        Then
        System.out.println("Potential_occupant_injury_rating5 fired");
        car.setpotential_occupant_injury_rating (“high”);
        update(Car);

rule Car_eligibility1
        when
        car: Car(potential_occupant_injury_rating == “high”, car_eligibility != “not
        eligible”)
        Then
        System.out.println("Car_eligibility1 fired");
        car.setcar_eligibility (“not eligible”);
        update(Car);

rule Car_eligibility2
        when
        car: Car(potential_occupant_injury_rating == “extremely high”,
        car_eligibility != “provisional”)

        Then
        System.out.println("Car_eligibility2 fired");
        car.setcar_eligibility (“provisional”);
        update(Car);
```

```
rule Car_eligibility3
        when
        car: Car(potential_theft_rating == “high”, car_eligibility != “provisional”)
        Then
        System.out.println("Car_eligibility3 fired");
        car.setcar_eligibility (“provisional”);
        update(Car);

rule Car_eligibility4
        when
        car: Car(car_housed_same_state_owner == false, car_eligibility !=
        “provisional”)
        Then
        System.out.println("Car_eligibility4 fired");
        car.setcar_eligibility (“provisional”);
        update(Car);

rule Car_eligibility5
        when
        car: Car(potential_occupant_injury_rating != “high”,
        potential_occupant_injury_rating != “extremely high”,potential_theft_rating
        != “high”, car_housed_same_state_owner == true, car_eligibility !=
        “eligible”)
        Then
        System.out.println("Car_eligibility5 fired");
        car.setcar_eligibility (“eligible”);
        update(Car);


rule Driver_age_category1
        when
        driver: Driver(age_category == “young”, training_certification == false,
        drivers_age_category != “young driver”)
        Then
        System.out.println("Driver_age_category1 fired");
        driver.setdrivers_age_category (“young driver”);
        update(Driver);

rule Driver_age_category2
        when
        driver: Driver(age_category == “young”, training_certification == true,
        drivers_age_category != eligible driver)
        Then
        System.out.println("Driver_age_category2 fired");
        driver.setdrivers_age_category (“eligible driver”);
        update(Driver);

rule Driver_age_category3
        when
        driver: Driver(age_category == “senior”, training_certification == false,
```

```
        drivers_age_category != “senior driver”)
        Then
        System.out.println("Driver_age_category3 fired");
        driver.setdrivers_age_category (“senior driver”);
        update(Driver);

rule Driver_age_category4
        when
        driver: Driver(age_category == “senior”, training_certification == true,
        drivers_age_category != “eligible driver”)
        Then
        System.out.println("Driver_age_category4 fired");
        driver.setdrivers_age_category (“eligible driver”);
        update(Driver);

rule Driver_age_category5
        when
        driver: Driver(age_category == “typical”, drivers_age_category != “eligible
        driver”)
        Then
        System.out.println("Driver_age_category5 fired");
        driver.setdrivers_age_category (“eligible driver”);
        update(Driver);

rule Age_category1
        when
        driver: Driver(sex == “male”, age < 25, age_category != “young”)
        Then
        System.out.println("Age_category1 fired");
        driver.setage_category (“young”);
        update(Driver);

rule Age_category2
        when
        driver: Driver(sex == “female”, age < 20, age_category != “young”)
        Then
        System.out.println("Age_category2 fired");
        driver.setage_category (“young”);
        update(Driver);

rule Age_category3
        when
        driver: Driver(sex == “male”, age > 70, age_category != “Senior”)
        Then
        System.out.println("Age_category3 fired");
        driver.setage_category (“Senior”);
        update(Driver);

rule Age_category4
        when
        driver: Driver(sex == “female”, age > 70, age_category != “senior”)
```

```
        Then
        System.out.println("Age_category4 fired");
        driver.setage_category (“senior”);
        update(Driver);

rule Age_category5
        when
        driver: Driver(sex == “male”, age > 19, age < 71, age_category !=
        “typical”)
        Then
        System.out.println("Age_category5 fired");
        driver.setage_category (“typical”);
        update(Driver);

rule Age_category6
        when
        driver: Driver(sex == “female”, age > 24, age < 71, age_category !=
        “typical”)
        Then
        System.out.println("Age_category6 fired");
        driver.setage_category (“typical”);
        update(Driver);

rule Training_certification1
        when
        driver: Driver(kind_of_certification == “from school”, training_certification
        != true)
        Then
        System.out.println("Training_certification1 fired");
        driver.settraining_certification (true);
        update(Driver);

rule Training_certification2
        when
        driver: Driver(kind_of_certification == “licensed driver training company”,
        training_certification != true)
        Then
        System.out.println("Training_certification2 fired");
        driver.settraining_certification (true);
        update(Driver);

rule Training_certification3
        when
        driver: Driver(kind_of_certification == “a senior citizin driver’s refresher”,
        training_certification != true)
        Then
        System.out.println("Training_certification3 fired");
        driver.settraining_certification (true);
        update(Driver);

rule Training_certification1
```

```
        when
        driver: Driver(kind_of_certification != "from school", kind_of_certification !=
        "licensed driver training company", kind_of_certification != "a senior citizin
        driver's refresher", training_certification != false)
        Then
        System.out.println("Training_certification1 fired");
        driver.settraining_certification (false);
        update(Driver);


rule Driving_record_category1
        when
        driver: Driver(history == "concited of a DUI", driving_record_category !=
        "high risk driver")
        Then
        System.out.println("Driving_record_category1 fired");
        driver.setdriving_record_category ("high risk driver");
        update(Driver);

rule Driving_record_category2
        when
        driver: Driver(history == " Number of accidents is greather then 2",
        driving_record_category != "high risk driver")
        Then
        System.out.println("Driving_record_category2 fired");
        driver.setdriving_record_category ("high risk driver");
        update(Driver);

rule Driving_record_category3
        when
        driver: Driver(history == "more then 3 moving voilations in the last 2
        years", driving_record_category != "high risk driver")
        Then
        System.out.println("Driving_record_category3 fired");
        driver.setdriving_record_category ("high risk driver");
        update(Driver);


rule Vehicle_policy_eligibility_score1
        when
        car: Car(car_eligibility == "not eligible")
        vehicle_insurence_policy: Vehicle_insurence_policy(score:eligibility_score)
        not  Controle ( flag == "Vehicle_policy_eligibility_score1")
        Then
        System.out.println("Vehicle_policy_eligibility_score1 fired");
        vehicle_insurence_policy.seteligibility_score (score + 100);
        Controle controle = new Controle();
        controle.setflag("Vehicle_policy_eligibility_score1");
        update(Vehicle_insurence_policy);
        insert(controle);
```

```
rule Vehicle_policy_eligibility_score2
        when
        car: Car(car_eligibility == "privisional")
        vehicle_insurence_policy: Vehicle_insurence_policy(score:eligibility_score)
        not Controle ( flag == "Vehicle_policy_eligibility_score2")
        Then
        System.out.println("Vehicle_policy_eligibility_score2 fired");
        vehicle_insurence_policy.seteligibility_score (score + 50);
        Controle controle = new Controle();
        controle.setflag("Vehicle_policy_eligibility_score2");
        update(Vehicle_insurence_policy);
        insert(controle);

rule Vehicle_policy_eligibility_score3
        when
        car: Car(car_eligibility == "eligible")
        vehicle_insurence_policy: Vehicle_insurence_policy(score:eligibility_score)
        not Controle ( flag == "Vehicle_policy_eligibility_score3")
        Then
        System.out.println("Vehicle_policy_eligibility_score3 fired");
        vehicle_insurence_policy.seteligibility_score (score + 0);
        Controle controle = new Controle();
        controle.setflag("Vehicle_policy_eligibility_score3");
        update(Vehicle_insurence_policy);
        insert(controle);

rule Vehicle_policy_eligibility_score4
        when
        driver: Driver(drivers_age_category == "young driver")
        vehicle_insurence_policy: Vehicle_insurence_policy(score:eligibility_score)
        not Controle ( flag == "Vehicle_policy_eligibility_score4")
        Then
        System.out.println("Vehicle_policy_eligibility_score4 fired");
        vehicle_insurence_policy.seteligibility_score (score + 30);
        Controle controle = new Controle();
        controle.setflag("Vehicle_policy_eligibility_score4");
        update(Vehicle_insurence_policy);
        insert(controle);

rule Vehicle_policy_eligibility_score5
        when
        driver: Driver(drivers_age_category == "eligible driver")
        vehicle_insurence_policy: Vehicle_insurence_policy(score:eligibility_score)
        not Controle ( flag == "Vehicle_policy_eligibility_score5")
        Then
        System.out.println("Vehicle_policy_eligibility_score5 fired");
        vehicle_insurence_policy.seteligibility_score (score + 0);
        Controle controle = new Controle();
        controle.setflag("Vehicle_policy_eligibility_score5");
        update(Vehicle_insurence_policy);
        insert(controle);
```

```
rule Vehicle_policy_eligibility_score6
        when
        driver: Driver(drivers_age_category == "senior driver")
        vehicle_insurence_policy: Vehicle_insurence_policy(score:eligibility_score)
        not Controle ( flag == "Vehicle_policy_eligibility_score6")
        Then
        System.out.println("Vehicle_policy_eligibility_score6 fired");
        vehicle_insurence_policy.seteligibility_score (score + 20);
        Controle controle = new Controle();
        controle.setflag("Vehicle_policy_eligibility_score6")
        update(Vehicle_insurence_policy);
        insert(controle);

rule Vehicle_policy_eligibility_score7
        when
        driver: Driver(driving_record_category == "high risk driver")
        vehicle_insurence_policy: Vehicle_insurence_policy(score:eligibility_score)
        not Controle ( flag == "Vehicle_policy_eligibility_score7")
        Then
        System.out.println("Vehicle_policy_eligibility_score7 fired");
        vehicle_insurence_policy.seteligibility_score (score + 100);
        Controle controle = new Controle();
        controle.setflag("Vehicle_policy_eligibility_score7");
        update(Vehicle_insurence_policy);
        insert(controle);

rule Vehicle_policy_eligibility_score8
        when
        driver: Driver(status_Client == "preferred")
        vehicle_insurence_policy: Vehicle_insurence_policy(score:eligibility_score)
        not Controle ( flag == "Vehicle_policy_eligibility_score8")
        Then
        System.out.println("Vehicle_policy_eligibility_score8 fired");
        vehicle_insurence_policy.seteligibility_score (score - 50 );
        Controle controle = new Controle();
        controle.setflag("Vehicle_policy_eligibility_score8");
        update(Vehicle_insurence_policy);
        insert(controle);

rule Vehicle_policy_eligibility_score9
        when
        driver: Driver(status_Client == "elite")
        vehicle_insurence_policy: Vehicle_insurence_policy(score:eligibility_score)
        not Controle ( flag == "Vehicle_policy_eligibility_score9")
        Then
        System.out.println("Vehicle_policy_eligibility_score9 fired");
        vehicle_insurence_policy.seteligibility_score (score - 100 );
        Controle controle = new Controle();
        controle.setflag("Vehicle_policy_eligibility_score9");
        update(Vehicle_insurence_policy);
```

```
        insert(controle);

rule Vehicle_policy_eligibility_score10
        when
        driver: Driver(status_Client == "normal")
        vehicle_insurence_policy: Vehicle_insurence_policy(score:eligibility_score)
        not Controle ( flag == "Vehicle_policy_eligibility_score10")
        Then
        System.out.println("Vehicle_policy_eligibility_score10 fired");
        vehicle_insurence_policy.seteligibility_score (score - 0 );
        Controle controle = new Controle();
        controle.setflag("Vehicle_policy_eligibility_score10");
        update(Vehicle_insurence_policy);
        insert(controle);


rule Client_segment1
        when
        vehicle_insurence_policy : Vehicle_insurence_policy (eligibility_score < 100)
        client : Client (client_segment != "eligible for insurance")
        Then
        System.out.println("Client_segment1 fired");
        client.setclient_segment ("eligible for insurance");
        update(client);

rule Client_segment2
        when
        vehicle_insurence_policy : Vehicle_insurence_policy (eligibility_score >=
        100, eligibility_score <= 250)
        client : Client (client_segment != "reviewed by manager")
        Then
        System.out.println("Client_segment2 fired");
        client.setclient_segment ("reviewed by manager");
        update(client);




rule Client_segment3
        when
        vehicle_insurence_policy : Vehicle_insurence_policy (eligibility_score > 250)
        client : Client (client_segment != "not eligible for insurance")
        Then
        System.out.println("Client_segment3 fired");
        client.setclient_segment ("not eligible for insurance");
        update(client);

rule Client_segment4
        when
        client: Client(userv_portfolio_for_15_years == true, client_segment !=
        "eligible for insurance")
```

```
        Then
        System.out.println("Client_segment4 fired");
        client.setclient_segment ("eligible for insurance");
        update(client);


rule Base_premium_for_each_car1
        when
        car: Car(type_of_car == "compact")
        vehicle_insurence_policy : Vehicle_insurence_policy (base_premium !=250)
        Then
        System.out.println("Base_premium_for_each_car1 fired");
        Vehicle_insurence_policy.setbase_premium (250);
        update(Vehicle_insurence_policy);


rule Base_premium_for_each_car2
        when
        car: Car(type_of_car == "sedan")
        vehicle_insurence_policy : Vehicle_insurence_policy (base_premium != 400)
        Then
        System.out.println("Base_premium_for_each_car2 fired");
        Vehicle_insurence_policy.setbase_premium (400);
        update(Vehicle_insurence_policy);


rule Base_premium_for_each_car3
        when
        car: Car(type_of_car == "luxury")
        vehicle_insurence_policy : Vehicle_insurence_policy (base_premium != 500)
        Then
        System.out.println("Base_premium_for_each_car3 fired");
        Vehicle_insurence_policy.setbase_premium (500);
        update(Vehicle_insurence_policy);

rule premium_increase_for_each_car1
        when
        car: Car(age_of_car == "model year is current year or next year")
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_car)
        not Controle ( flag == "premium_increase_for_each_car1")
        Then
        System.out.println("premium_increase_for_each_car1 fired");
        vehicle_insurence_policy.set premium_increase_for_each_car (score +
        400);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_car1");
        update(Vehicle_insurence_policy);
        insert(controle);
```

```
rule premium_increase_for_each_car2
        when
        car: Car(age_of_car == "model year is within past 4 years")
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_car)
        not Controle ( flag == "premium_increase_for_each_car2")
        Then
        System.out.println("premium_increase_for_each_car2 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_car (score + 300);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_car2");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_increase_for_each_car3
        when
        car: Car(age_of_car == "model year is between past 4 years and past 10
        years")
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_car)
        not Controle ( flag == "premium_increase_for_each_car3")
        Then
        System.out.println("premium_increase_for_each_car3 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_car (score + 250);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_car3");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_increase_for_each_car4
        when
        vehicle_insurence_policy:
        Vehicle_insurence_policy(uninsured_morerist_coverage == true , score:
        premium_increase_for_each_car)
        not Controle ( flag == "premium_increase_for_each_car4")
        Then
        System.out.println("premium_increase_for_each_car4 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_car (score + 600);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_car4");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_increase_for_each_car5
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(medical coverage ==
        true , score: premium_increase_for_each_car)
        not Controle ( flag == "premium_increase_for_each_car5")
        Then
        System.out.println("premium_increase_for_each_car5 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_car (score + 600);
```

```
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_car5");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_increase_for_each_car6
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_car)
        car: Car (potential_occupant_injury_rating == " extremely high")
        not Controle ( flag == "premium_increase_for_each_car6")
        Then
        System.out.println("premium_increase_for_each_car6 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_car (score +
        1000);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_car6");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_increase_for_each_car7
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_car)
        car: Car (potential_occupant_injury_rating == "high")
        not Controle ( flag == "premium_increase_for_each_car7")
        Then
        System.out.println("premium_increase_for_each_car7 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_car (score + 500);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_car7");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_increase_for_each_car8
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_car)
        car: Car (potential_theft_rating == "high")
        not Controle ( flag == "premium_increase_for_each_car8")
        Then
        System.out.println("premium_increase_for_each_car8 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_car (score + 500);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_car8");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_increase_for_each_car9
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
```

```
        premium_increase_for_each_car)
        car: Car (car_housed_same_state_owner == false)
        not Controle ( flag == "premium_increase_for_each_car9")
        Then
        System.out.println("premium_increase_for_each_car9 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_car (score +
        1000);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_car9");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_discount_for_each_car1
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_car)
        car: Car (type_of_airbag == "driver's airbag")
        not Controle ( flag == "premium_discount_for_each_car1")
        Then
        System.out.println("premium_discount_for_each_car1 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_car (score + 12 );
        Controle controle = new Controle();
        controle.setflag("premium_discount_for_each_car1");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_discount_for_each_car2
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_car)
        car: Car (type_of_airbag == "driver's and front passenger airbag")
        not Controle ( flag == "premium_discount_for_each_car2")
        Then
        System.out.println("premium_discount_for_each_car1 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_car (score + 15 );
        Controle controle = new Controle();
        controle.setflag("premium_discount_for_each_car2");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_discount_for_each_car3
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_car)
        car: Car (type_of_airbag == "driver's front passenger and side panel
        airbag")
        not Controle ( flag == "premium_discount_for_each_car3")
        Then
        System.out.println("premium_discount_for_each_car3 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_car (score + 18 );
        Controle controle = new Controle();
```

```
        controle.setflag("premium_discount_for_each_car3");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_discount_for_each_car4
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_car)
        car: Car (potential_theft_rating == "high", alarm == true)
        not Controle ( flag == "premium_discount_for_each_car4")
        Then
        System.out.println("premium_discount_for_each_car4 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_car (score + 10 );
        Controle controle = new Controle();
        controle.setflag("premium_discount_for_each_car4");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_increase_for_each_driver1
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_driver)
        driver: Driver (age_category == "young", marital_status == "married",
        Located_in_CA_NY_or_VA == true)
        not Controle ( flag == "premium_increase_for_each_driver1")
        Then
        System.out.println("premium_increase_for_each_driver1 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_driver (score +
        700);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_driver1");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_increase_for_each_driver2
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_driver)
        driver: Driver (age_category == "young", marital_status == "single",
        Located_in_CA_NY_or_VA == true)
        not Controle ( flag == "premium_increase_for_each_driver2")
        Then
        System.out.println("premium_increase_for_each_driver2 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_driver (score +
        720);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_driver2");
        update(Vehicle_insurence_policy);
        insert(controle);
```

```
rule premium_increase_for_each_driver3
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_driver)
        driver: Driver (age_category == "young", marital_status == "married",
        Located_in_CA_NY_or_VA == false)
        not Controle ( flag == "premium_increase_for_each_driver3")
        Then
        System.out.println("premium_increase_for_each_driver3 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_driver (score +
        300);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_driver3");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_increase_for_each_driver4
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_driver)
        driver: Driver (age_category == "young", marital_status == "single",
        Located_in_CA_NY_or_VA == false)
        not Controle ( flag == "premium_increase_for_each_driver4")
        Then
        System.out.println("premium_increase_for_each_driver4 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_driver (score +
        300);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_driver4");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_increase_for_each_driver5
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_driver)
        driver: Driver (age_category == "senior", Located_in_CA_NY_or_VA= true)
        not Controle ( flag == "premium_increase_for_each_driver5")
        Then
        System.out.println("premium_increase_for_each_driver5 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_driver (score +
        500);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_driver5");
        update(Vehicle_insurence_policy);
        insert(controle);
```

```
rule premium_increase_for_each_driver6
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_driver)
        driver: Driver (age_category == "senior", Located_in_CA_NY_or_VA= false)
        not Controle ( flag == "premium_increase_for_each_driver6")
        Then
        System.out.println("premium_increase_for_each_driver6 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_driver (score +
        200);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_driver6");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_increase_for_each_driver7
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_driver)
        driver: Driver (driving_record_category == "high risk driver")
        not Controle ( flag == "premium_increase_for_each_driver7")
        Then
        System.out.println("premium_increase_for_each_driver7 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_driver (score + 0);
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_driver7");
        update(Vehicle_insurence_policy);
        insert(controle);

rule premium_increase_for_each_driver8
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        premium_increase_for_each_driver)
        driver: Driver (driving_record_category == "high risk driver")
        not Controle ( flag == "premium_increase_for_each_driver8")
        Then
        System.out.println("premium_increase_for_each_driver8 fired");
        vehicle_insurence_policy.setpremium_increase_for_each_driver (score +
        (accidents * 150));
        Controle controle = new Controle();
        controle.setflag("premium_increase_for_each_driver8");
        update(Vehicle_insurence_policy);
        insert(controle);
```

```
rule market_segment_discount1
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        market_segment_discount)
        client: Client (status_Client == "preferred")
        not Controle ( flag == "market_segment_discount1")
        Then
        System.out.println("market_segment_discount1 fired");
        vehicle_insurence_policy.setmarket_segment_discount (score + 250);
        Controle controle = new Controle();
        controle.setflag("market_segment_discount1");
        update(Vehicle_insurence_policy);
        insert(controle);

rule market_segment_discount2
        when
        vehicle_insurence_policy: Vehicle_insurence_policy(score:
        market_segment_discount)
        client: Client (status_Client == "elite")
        not Controle ( flag == "market_segment_discount2")
        Then
        System.out.println("market_segment_discount2 fired");
        vehicle_insurence_policy.setmarket_segment_discount (score + 500);
        Controle controle = new Controle();
        controle.setflag("market_segment_discount2");
        update(Vehicle_insurence_policy);
        insert(controle);
```

Bijlage 5

```
public class Client {
	private Boolean in_familyportfolio;
	private String status_Client;
	private int number_products_in_portfolio;
	private int dollars_in_family_combined_balance;
	private int dollars_in_personal_combined_balance;
	private int products_in_familyportfolio;
	private String client_segment;
	private boolean userv_portfolio_for_15_years;


public Client(Boolean in_familyportfolio, String status_Client, int
number_products_in_portfolio, int dollars_in_ family_combined_balance, int
products_in_familyportfolio, String client_segment, boolean
userv_portfolio_for_15_years){
	this.in_familyportfolio = in_familyportfolio;
	this.status_client = status_client;
	this.number_products_in_portfolio = number_products_in_portfolio;
	this.dollars_in_ family_combined_balance = dollars_in_
	family_combined_balance;
	this.products_in_familyportfolio = products_in_familyportfolio;
	this.dollars_in_personal_combined_balance =
	dollars_in_personal_combined_balance;
	this.client_segment = client_segment;
	this.userv_portfolio_for_15_years = userv_portfolio_for_15_years;
		}


public class Car {
	private boolean convertible;
	private int price;
	private Boolean on_high_theft_probability_list;
	private String potential_occupant_injury_rating;
	private String potential_theft_rating;
	private String type_of_airbag;
	private Boolean Roll_bar;
	private String car_eligibility;
	private Boulean car_housed_same_state_owner;
	private String type_of_car;
	private int age_of_car;
	private Boolean alarm;
	private int year;
	private String make_and_model;


public Car (boolean convertible, int price, Boolean on_high_theft_probability_list,
String potential_occupant_injury_rating, String potential_theft_rating, String
```

```
type_of_airbag, Boolean Roll_bar, String auto_eligibility, Boulean
car_housed_same_state_owner, String type_of_car, int age_of_car, Boolean alarm,
int year, String make_and_model){
        this.convertible = convertible;
        this.price = price;
        this.on_high_theft_probability_list = on_high_theft_probability_list;
        this.potential_occupant_injury_rating = potential_occupant_injury_rating
        this.potential_theft_rating = potential_theft_rating;
        this.type_of_airbag = type_of_airbag;
        this.Roll_bar = Roll_bar;
        this.auto_eligibility = auto_eligibility;
        this.car_housed_same_state_owner = car_housed_same_state_owner;
        this.type_of_car = type_of_car;
        this.age_of_car = age_of_car;
        this.alarm = alarm;
        this.year = year;
        this.make_and_model = make_and_model;



public class Driver {
        private String age_category;
        private boolean training_certification;
        private String driving_record_category;
        private int accidents;
        private String drivers_age_category;
        private int age;
        private String sex;
        private String marital_status;
        private Boolean Located_in_CA_NY_or_VA;
        private String kind_of_certification;
        private String history;


}

public Driver(String age_category, boolean training_certification, String
driving_record_category, int accidents, String drivers_age_category, int age, String
sex, String marital_status, Boolean Located_in_CA_NY_or_VA, String
kind_of_certification, String history){
        this.age_category = age_category;
        this.training_certification = training_certification;
        this.driving_record_category = driving_record_category;
        this.accidents = accidents;
        this.drivers_age_category = drivers_age_category;
        this.age = age;
        this.sex = sex;
        this.marital_status = marital_status;
        this.Located_in_CA_NY_or_VA = Located_in_CA_NY_or_VA;
        this.kind_of_certification = kind_of_certification;
```

```
        this.history = history;

        }



public class Vehicle_insurence_policy {
        private int eligibility_score
        private String client_eligibility_for_insurance
        private int base_premium
        private Boolean medical coverage
        private boolean uninsured_morerist_coverage
        private int premium_increase_for_each_car
        private int premium_discount_for_each_car
        private int premium_increase_for_each_driver
        private int market_segment_discount;
}

public Vehicle_insurence_policy (int eligibility_score, String
client_eligibility_for_insurance, int base_premium, Boolean medical coverage,
boolean uninsured_morerist_coverage, int premium_increase_for_each_car, int
premium_discount_for_each_car, int premium_increase_for_each_driver, int
market_segment_discount){

        this.eligibility_score = eligibility_score;
        this.client_eligibility_for_insurance = client_eligibility_for_insurance;
        this.base_premium = base_premium;
        this.medical coverage = medical coverage;
        this.uninsured_morerist_coverage = uninsured_morerist_coverage;
        this.premium_increase_for_each_car = premium_increase_for_each_car;
        this.premium_discount_for_each_car = premium_discount_for_each_car;
        this.premium_increase_for_each_driver =
premium_increase_for_each_driver;
        this.market_segment_discount = market_segment_discount;
        }



public class Controle {
        private String flag;
}
public Controle (string flag){


        this.flag = flag;
}
```

Bijlage 6

```
Waarvan wil u de premie berekenen?
A : een auto
B : een client
C : een client en een auto
C
Is uw auto een convertible Ja/Neen
Neen
Wat is de Prijs van uw auto? ($ niet toevoegen)
23000
Welke type airbag heeft u?
A : driver's airbag?
B : driver's and front passenger airbag
C : driver's front passenger and side panel airbag?
D : Geen
B
Heeft u een rollbar Ja/Neen
Neen
Is de auto in dezelfde staat ingeschreven als de eigenaar? Ja/Neen
Ja
Wat is het type van de auto
A : compact
B : sedan
C : luxury
D : andere
C
Wat is het bouwjaar van uw auto?
A : 2011 of 2012
B : 2010, 2009, 2008 of 2007
C : 2006, 2005, 2004, 2003 of 2002
D : andere
B
Heeft de auto een alarm?  Ja/Neen
Ja
Is de auto gebouwd tussen 1964 en 1970? Ja/Neen
Neen
Wat is het merk van de auto?
A : VW Bug
B : Porsche
C : BMW
D : andere
D
Maakt de client deel uit van een familie portfolio? Ja/Neen
Neen
```

```
Hoeveel producten heeft de client in zijn portfolio?
3
Hoeveel dollars zitten in de balans van de persoonlijke portfolio? (
geen $ plaatsen)
44000
Heeft de client reeds een Userv portfolio voor minstens 15jaar Ja/Neen
Neen
Hoeveel ongelukken heeft de client reeds gehad?
0
Hoe oud is de client?
34
wat is het geslacht van de client? A:Man, B:Vrouw
A : Man
B : Vrouw
A
wat is we wettelijke status van de client?  A: getrouwd, B niet
getrouwd
A: getrouwd
B: niet getrouwd
A
Woont de client in CA, NY of VA?   Ja/Neen
Neen
Welke training heeft u gehad?
A : Op school
B : Opfrissingscursus oor senioren
C : Van een erkende rijschool
D : andere of geen
D
Welk van onderstaand verleden geldt voor de client?
A : DUI gepleegd
B : meer dan 2 ongelukken gehad
C : meer dan 3 overtreden gehad in de laatste 2 jaar
D : geen van bovenstaande
D
Wil de client de medische kosten dekken deze de verzekering?   Ja/Neen
Ja
Wil men verzekering voor de onverzekerde bestuurders ?   Ja/Neen
Ja
premium_discount_for_each_car1 fired
premium_increase_for_each_car2 fired
Base_premium_for_each_car3 fired
Client_segment1 fired
premium_increase_for_each_car5 fired
premium_increase_for_each_car4 fired
StatusClient1 fired
market_segment_discount1 fired
Car_eligibility5 fired
```

```
Vehicle_policy_eligibility_score3 fired
Potential_theft_category4 fired
Potential_occupant_injury_rating3 fired
-------------------------------------
-------------------------------------
De client is eligible for insurance
-------------------------------------
De basispremie voor de auto van deze client is $500
De basispremie wordt voor deze auto verhoogd met $1500
De basispremie wordt voor deze auto verlaagd met  15%
De basis premie wordt per bestuurder verhoogd met $0
De status van deze client is preferred
Op basis van deze status wordt de basispremie voor deze client
verlaagd met $250
```

Bijlage 7

```
*** Decision PremiumDecision ***
Decision has been initialized
Decision PremiumDecision: Status Client
Conclusion: Status_client Is Preferred
Decision PremiumDecision: Potential Theft Category
Decision PremiumDecision: On High Theft Probability Auto List
Conclusion: On_high_theft_probability_list Is No
Decision PremiumDecision: Potential Occupant Injury Category
Conclusion: Potential_occupant_injury_rating Is Moderate
Decision PremiumDecision: Auto Eligibility
Decision PremiumDecision: Drivers Age Category
Decision PremiumDecision: Age Category
Conclusion: Age_category Is Typical
Decision PremiumDecision: Training Certification
Conclusion: Training_certification Is No
Decision PremiumDecision: Driving Record Category
Conclusion: Driving_record_category Is No
Decision PremiumDecision: Eligibility Score
Conclusion: Eligibility_score Is -50
verlagen met 50 from EligibilityScore
Decision PremiumDecision: Client Segment
Conclusion: Client_segment Is Eligible_for_insurance
Decision PremiumDecision: Base Premium For Each Car
Conclusion: Base_premium Is 400
DE BASISPREMIE IS $400 from BasePremiumForEachCar
Decision PremiumDecision: Premium Increase For Each Car
Conclusion: Premium_increase_for_each_car Is 300
VERHOGEN MET $300 from PremiumIncreaseForEachCar
Decision PremiumDecision: Premium Discount For Each Car
Conclusion: Premium_discount_for_each_car Is 15
% VERHOGEN MET 15% from PremiumDiscountForEachCar
Decision PremiumDecision: Premium Increase For Each Driver
Conclusion: Premium_increase_for_each_driver Is 150
PREMIE VERHOGEN MET $150 PER ONGELUK from PremiumIncreaseForEachDriver
Decision PremiumDecision: Market Segment Discount
Conclusion: Market_segment_discount Is 250
DEZE KLANT HEEFT STATUS "PREFERRED" EN KRIJGT DAAROM EEN KORTING OP DE
PREMIE VAN 250 $ from MarketSegmentDiscount
Decision PremiumDecision: *** OpenRules made a decision ***
```

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Modelleren van processen en beslissingen : een geval studie

Richting: **master in de toegepaste economische wetenschappen: handelsingenieur in de beleidsinformatica**
Jaar: **2012**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Verdeyen, Natasja

Datum: **9/01/2012**