# De toepasbaarheid van Rule engines voor Sogyo

Bachelor scriptie

Tim de Jager (1519215)

1 juni 2010

Begeleiders:
Erik Mulder
Peter van Rooijen

Hogeschool Utrecht
Faculteit Natuur en Techniek

# Voorwoord

Dit document is geschreven in kader van mijn afstudeeropdracht van de opleiding Informatica. Deze scriptie is het resultaat van de afstudeeropdracht die in de afgelopen maanden is uitgevoerd.

De afstudeeropdracht was erg intressant en uitdagend, ik heb gedurende mijn afstudeerperiode een hoop geleerd hier ben ik erg blij om. Bij mijn afstudeerbedrijf Sogyo heb ik genoten van een prettige tijd en ik ben erg onder de indruk hoe open en vriendelijk iedereen bij Sogyo is, deze goede werksfeer is zeker bevorderend voor de productiviteit. Mijn afstudeeropdracht had niet succesvol afgerond kunnen worden zonder de hulp van een aantal personen, deze wil ik graag nog bedanken.

Allereerst wil ik Peter van Rooijen en Erik Mulder bedanken voor de begeleiding en inspiratie, ik ben altijd erg onder de indruk geweest van jullie enthousiasme en bereidheid om te helpen. Graag wil ik ook mijn familie bedanken, die mij de kans hebben gegeven deze opleiding te volgen en mij altijd hebben gesteund. Jos van Egmond en Ben d'hont, jullie hebben mijn al plezierige afstudeerperiode nog plezierig gemaakt. Ook wil ik graag Paul Visser en Tjeerd Broersma bedanken, jullie hebben met een kritisch oog naar mijn scriptie gekeken. Niets weerhield jullie van jullie zoektoch naar fouten en verbeteringen.

Ten slotte, wil ik Sogyo nog bedanken. Voor alle hulp en tips van collega's en de plezierige tafeltennis wedstrijden tijdens de lunch.

# Managementsamenvatting

Rule engines zijn een interessant onderwerp. Veel ontwikkelaars hebben wel van een Rule engine gehoord, maar weten niet precies wat het is [16]. Ze zijn vaak al wel bekend met Prolog een logische programmeertaal, deze programmeertaal wordt echter niet in alledaagse situaties gebruikt. Rule engines proberen daarentegen wel in deze situaties toepasbaar te zijn. De vraag blijft aanwezig, of een Rule engine toepasbaar is voor gebruik binnen een organisatie.

## Doel

Sogyo, een software en detacheringsbedrijf gevestigd in De Bilt, wilde deze vraag graag beantwoord hebben. Door Sogyo wordt veel onderzoek gedaan naar nieuwe ontwikkelingen op software gebied. Onderzoek naar Rule engines was nog nooit gedaan door Sogyo, dus werd besloten een afstudeerproject te starten. Het doel van dit afstudeerproject was om de toepasbaarheid van een Rule engine voor Sogyo te bepalen alsmede een geschikt Rule engine product vinden.

## Opzet

Tijdens het project is besloten om eerst een geschikt Rule engine product te vinden, dat aan bepaalde eisen voldoet. Deze eisen waren zowel specifieke eisen van Sogyo als algemene en technische eisen. Met dit uitgekozen product is vervolgens een Proof of Concept opgezet. Het Proof of Concept maakte als casus gebruik van een project uitgevoerd door Sogyo en een ontwikkelmethodiek genaamd Domain Driven Design. Deze methodiek wordt veelvoudig gebruikt door Sogyo. Zodoende kon het Proof of Concept een goed beeld geven over de toepasbaarheid van Rule engines voor Sogyo.

Het project bestond uit twee fase:

- Onderzoeksfase. In deze fase is onderzoek gedaan naar de theorie achter Rule engines. Verder is er een iventarisatie gemaakt van de producten waarvan een aantal aan tests zijn onderworpen. Als resultaat is er een aanbeveling voor een product uitgekomen.
- Implementatiefase. In deze fase is het Proof of Concept uitgewerkt met behulp van het uitgekozen product uit de onderzoeksfase. In deze fase kwamen Domain Driven Design, Rule engines en een project van Sogyo in de praktijk samen. In deze fase zijn ook een aantal conclusie getrokken

## Conclusie

De volgende conclusie zijn getrokken uit het project:

- Het Rule engine product wat momenteel het meest aan de eisen voldoet is JBoss Drools.
- Rule engines zijn alleen geschikt voor specifieke situaties.
- Rule engines zijn voor project Norma niet goed geschikt.
- Rule engines en Domain Driven Design, kan gecombineerd worden mits op de juiste wijze wordt gedaan.
- Rule engines zijn in theorie toepasbaar voor Sogyo, echter kan alleen gebruik worden gemaakt van een Rule engine indien domeinmodel zich daarvoor leent.

# Begrippenlijst

**Activatie van Regels** Het activeren van Regels beteken dat de Regel op de agenda wordt geplaats. Deze regel kan er nog van af worden gehaald indien de feiten veranderen. Activatie is in principe het 'klaar zetten' van regels.

**Aggregate Root** Een begrip in Domain Driven Design, encapsuleert verschillende aan elkaar gerelateerde entiteiten.

**Backward Chaining** Backward chaining is een algortime gebruikt door Rule engines. In dit algoritme wordt uitgegaan van de conclusies en wordt naar de predicaten toegwerkt. Backward chaining wordt gebruikt door bijv. Prolog.

**CLISP** Een Rule engine uit de eind jaren 80. De taal gebruikt om met het systeem aan te sturen was LISP.

**Domain Driven Design (DDD)** Domain Driven Design is een ontwikkelmethodie waar het domein centraal gesteld wordt. Domain Driven Design probeert een 'dun' domein te vermijden, de bedrijfslogica moet gencapsuleert worden in het domein. De grondleger van deze methodiek is Eric Evans. Domain Driven Design wordt regelmatig afgekort naar DDD.

**Domain Specific Language (DSL)** Een Domain Specific Language is een taal gemaakt om in een specifiek domein gebruikt te worden. Shaders gebruikt voor het programmeren van video kaarten zijn hier een voorbeeld van. Domain Specific Langauges zijn vaak niet 'Turing complete'.

**Drools** Een Rule engine product van JBoss, dit product is voor Sogyo uitgekozen als een geschikt product.

**Entity/Entiteit** En begrip in Domain Driven Design, duidt objecten aan waarvan de identiteit moet worden bijgehouden.

**Expert system** Een systeem wat tracht de kennis van experts na te bootsen. Rule engines zijn regel gebaseerde expertsystemen.

**Feit/Fact** In de context van een Rule engine is een feit een object met bepaalde eigenschappen, deze eigenschappen bepalen welke regels uitgevoerd moeten worden.

**Forward Chaining** Forward chaining is een algoritme gebruikt door Rule engines. In dit algortime wordt uitgegaan van de predicaten en wordt naar de conclusies toegewerkt. Forward chaining wordt i.p.v. backward chaining, gebruikt door de onderzocht Rule engines.

**Hogeschool Utrecht** Dit is de HBO instelling waar de afstudeerder zijn opleiding heeft gevolgd.

**Inference Engine** Het brein van de Rule engine. Redeneert over de regels en de feiten.

**Knowledge Base** Bevat de knowledge/kennis van een Rule engine of Expert system.

**Lisp** Een functionele programmeer taal bedacht door John McCarthy, voor het eerst gezien in 1958.

**Norma** Een intern project uitgevoerd door Sogyo, dit project is als casus gebruikt voor het Proof of Concept.

**Prolog** Een logische programmeer taal uit 1972. Prolog is gebaseerd op dezelfde principes als een Rule engine, echter is Prolog een 'general purpose language'.

**Proof of Concept** Een Proof of Concept levert bewijs of een Concept toepasbaar is.

**Read Eval Print Loop (REPL)** Deze term wordt vaak gebruikt in combinatie met Lisp. De REPL is een interactieve omgeving waar code in uitgevoerd kan worden. De REPL is ook terug te vinden in modernere talen zoals bijvoorbeeld Python en Ruby.

**Regels/Rules** Regels in de context van een Rule engine is een declaratie die aan de hand van een Predicaat een bepaalde conclusie uitvoert. Regels worden geactiveerd door feiten.

**Rete Agenda** Bepaalt in welke volgorde de Regels uitgevoerd moeten worden.

**Rule engine** Het onderwerp van de afstudeeropdracht. Rule engines zijn systemen waar regels in gedeclareerd kunnen worden. De Rule engine redeneert aan de hand van regels over het systeem.

**Shortlist** Een lijst afgeleid van een grotere lijst, waar de elementen zijn wegelaten die niet meer relevant zijn of aan de eisen voldoen.

**Sogyo** Dit is het bedrijf waarvoor de afstudeeropdracht wordt uitgevoerd.

**Standard Widget Toolkit (SWT)** SWT is een GUI library, een alternatief voor Swing.

**Uitvoer van Regels** Het daadwerkelijke uitvoeren van Regels, zodra de regels zijn uitgevoerd zijn de consequenties vaak onomkeerbaar. De uitvoer van Regels kan resulteren in nieuwe activaties.

**Ubiquitous Language** In Domain Driven Design is dit de alomvattende taal. Deze taal word gebruikt om over het domein te communiceren. Deze taal dient terug te komen in het model.

**Value Object** Een begrip in Domain Driven Design, duidt een object aan waarvan de identiteit niet hoeft worden bijgehouden.

# Inhoudsopgave

# Deel I.

# Inleiding

# 1. Inleiding

Rule engines zijn een interessant onderwerp. Veel ontwikkelaars hebben wel van een Rule engine gehoord, maar weten niet precies wat het is [16]. Ze zijn vaak al wel bekend met Prolog, een logische programmeertaal, deze programmeertaal wordt echter niet in alledaagse situaties gebruikt. Rule engines proberen daarentegen wel in deze situaties toepasbaar te zijn. De vraag blijft aanwezig, of een Rule engine toepasbaar is voor gebruik binnen een organisatie.

## 1.1. Rule engine voor Sogyo

Binnen Sogyo worden regelmatig onderzoeken gestart over verscheidene onderwerpen. Deze onderzoeken worden uitgevoerd door medewerkers die momenteel niet gedetacheerd zijn. Sogyo probeert met deze onderzoeken de ontwikkelingen in software markt bij te blijven. In het verleden zijn er projecten geweest waar Sogyo verwacht dat een Rule engine eventueel nuttig had kunnen zijn, dit waren projecten met complexe bedrijfslogica. Aangezien Sogyo weinig kennis heeft op het gebied van Rule engines, is besloten een afstudeerder zich te laten verdiepen in dit onderwerp. Deze afstudeerder kan vervolgens, na afloop van het project, een advies geven aan Sogy omtrent het gebruik van Rule engines en de toepasbaarheid ervan voor Sogyo.

## 1.2. Rule engine in vogelvlucht

Wat een Rule engine precies is wordt later in de scriptie in detail behandeld. Het is echter nuttig om in grote lijnen een beeld te hebben over de functie en het nut van een Rule engine. Een Rule engine is een systeem dat probeert de kennis van domeinexperts te modelleren. Dit doet een Rule engine aan de hand van regels. In elke bedrijfsector bestaan 'businessregels', deze regels zorgen dat er beslissingen worden gemaakt over uiteenlopende onderwerpen: klanten, producten, patiënten etc. Een Rule engine probeert deze regels te structureren zodat de software deze beslissingen kan maken en de consequenties kan overzien.

### 1.2.1. Welk probleem lost het op?

Software ontwikkelaars weten vaak niet veel van het domein af maar wel van het bouwen van software, Domeinexperts daarentegen weten niet veel van software af maar wel van het domein. Een Rule engine probeert hier tussen te gaan zitten, het geef ontwikkelaars de mogelijkheid om de kennis van domeinexperts op een natuurlijke wijze te kunnen

modelleren. Vele Rule engines bevatten tools voor domeinexperts om regels op te stellen. Een Rule engine probeert op een logischere, duidelijker en overzichtelijkere wijze regels te modelleren dan een traditionele softwaretaal dit doet.

## 1.3. Leeswijzer

De scriptie is opgedeelt in drie delen: Inleiding, Opdrachtanalyse en de Conclusie.

**Inleiding** Begint met deze inleiding. Daarna wordt de afstudeeropdracht, Sogyo en Domain Driven Design besproken.

**Opdrachtanalyse** Begint met een inleiding over Rule engines. Daarna wordt het onderzoek behandeld en de verschillende fase's van het onderzoek. Ten slotte wordt de implementatiefase en het Proof of Concept in detail behandeld.

**Conclusie** Begint met een korte samenvatting over de verschillend fases. Daarna volgt de conclusie. Ten slotte wordt er advies gegeven aan Sogyo.

### 1.3.1. Referenties

De drie delen zijn opgebroken in hoofdstukken, deze hoofdstukken bestaan uit secties die uit subsecties en paragrafen bestaan. Referenties aangeduid met cijfers referen naar delen van dit document. Referenties aangeduid met letters refereren naar de bijlage. De bronnen van citaten zijn terug te vinden in de bibliografie met het bijbehorende cijfer weergegven in blokhaken.

# 2. Achtergrond

## 2.1. Bedrijf

Het bedrijf waarvoor de afstudeeropdracht is uitgevoerd heet Sogyo. Sogyo zegt over zichzelf:

> "Sogyo, opgericht in 1995, is een innovatieve, creatieve
> en vooral inhoudelijk gedreven IT-dienstverlener. Sogyo ontwikkelt IT-toepassingen, -talent en –organisaties, door middel van de volgende diensten:
>
> - Detachering & Consultancy: inzet van (technisch) consultants en software developers in de volle breedte van het veld. Kundige experts met passie voor hun vak
> - Opleidingen: ruime ervaring met het ontwikkelen van IT-talent door verbreding en verdieping van een breed scala aan competenties. Zowel voor eigen medewerkers als in-company
> - Development: uitgebreide ervaring in het realiseren van maatwerk projecten voor onze klanten"
>
> [13]

Sogyo heeft mij ingehuurd als afstudeerder en ik heb de afstudeeropdracht uitgevoerd op de development afdeling. De opdracht was individueel, ik ben tijdens de opdracht wel begeleid vanuit Sogyo.

Dit is het eerste project binnen Sogyo wat gebruik heeft gemaakt van Rule engines. De opdrachtgever heeft aangegeven onderzoek te willen doen naar de toepasbaarheid van Rule engines binnen Sogyo. In voorgaande projecten was er eventueel sprake dat een Rule engine nuttig zou zijn, deze mogelijkheid willen ze nu graag onderzocht hebben.

### 2.1.1. Rolverdeling

De rolverdeling is als volgt:

- De uitvoerende partij: Tim de Jager, de afstudeerder. Deze zal het project onder toezicht van de opdrachtgever uitvoeren.
- De opdrachtgever: Sogyo. Deze heeft de opdracht samen met mij (de afstudeerder) geformuleerd. Er is een begeleider vanuit Sogyo toegewezen namelijk Erik Mulder.

- De toezichthouder: de Hogeschool Utrecht. Deze zal gedurende het project de status peilen en uiteindelijk ook de opdracht beoordelen. Ook hier is een begeleider toegewezen namelijk Peter van Rooijen.

# 3. Opdracht

## 3.1. Inleiding

Mijn opdracht is onderzoek te doen naar het gebruik van Rule engines voor Sogyo en een implementatie te bouwen als een 'Proof of Concept'. Er moet dus gekeken worden naar de verschillende mogelijkheden van een Rule engine en welke Rule engine in de Sogyo ontwikkelmethodiek zou passen. Omdat er vanuit Sogyo niet veel kennis over het onderwerp is, is het belangrijk deze te vergaren. Het is ook van belang de wensen van Sogyo te inventariseren. Zodoende kan er een advies worden gegeven over een te gebruiken product.

### 3.1.1. Fases

Het project kent in principe twee globale fases, die wederom zijn opgesplitst in kleinere onderdelen:

1. Onderzoeksfase. In deze fase wordt onderzoek gedaan naar het gebruik van Rule engines.
2. Implementatiefase. In deze fase wordt het Proof of Concept ontwikkeld.

Door middel van het Proof of Concept moet aantoonbaar worden gemaakt wat het daadwerkelijke verschil is tussen het gebruik van een Rule engine t.o.v. een traditionele implementatie. Zo kan duidelijk worden wat het gebruik van een Rule engine oplevert. Ten slotte is het belangrijk de eigen ervaringen toe te kunnen lichten.

## 3.2. Doelstelling

Sogyo wil graag een beter inzicht krijgen in het gebruik van een Rule engine. In bepaalde gevallen, met name waarbij de bedrijfslogica aan veel verandering onderhevig is, of er een duidelijke vorm van afzonderlijke regels te specificeren is, biedt een Rule engine wellicht voordeel. Sogyo is daarom geïnteresseerd in de mogelijkheden van het integreren van Rule engines in hun bestaande ontwikkelmethodiek.

Hierbij moet gedacht worden aan het gemak waarmee Sogyo ontwikkelaars het product zouden kunnen gebruiken, evenals de aansluiting met de Domain Driven Design methodiek (zie hiervoor 3.5) die binnen Sogyo wordt toegepast.

Sogyo zou de verschillen graag waarneembaar willen krijgen d.m.v. een 'Proof of Concept'.

## 3.3. Onderzoek

Het onderzoek is bedoeld om de onderzoeksvraag (te lezen in sectie 6.1.1) te beantwoorden. Waar dit op neer komt is evalueren welk product het meest geschikt is voor Sogyo. Evenals een aantal vragen kunnen beantwoorden omtrent het gebruik van een Rule engine, in het algemeen en voor Sogyo specifiek.

## 3.4. Implementatie

De implementatie is een Proof of Concept. Dit Proof of Concept moet inzicht verschaffen over het feit of een Rule engine geschikt is voor gebruik binnen Sogyo.

## 3.5. Domain Driven Design

Domain Driven Design (DDD) is een methodiek voor het ontwikkelen van software, deze methodiek wordt gebruikt binnen Sogyo. Domain Driven Design is zelf zo uitgebreid dat het niet gewenst is om alle aspecten ervan te bespreken, hieronder volgt een korte introductie.

Domain Driven Design is een manier om de complexiteit aan te pakken die inherent is aan elk middelmatig tot groot software project. Het boek: “Domain-Driven Design, Tackling Complexity in the Heart of Sofware” is het eerste boek waarin dit onderwerp behandelt werd. Het boek, wat geschreven is door Eric Evans, beschrijft een wijze waarop het domein kan worden gestructureerd. Volgens Kyle Brown, een auteur van verschillende Software boeken, gebruikt Eric Evans een manier van ontwerpen dat al langer door ‘experienced object designers’ wordt toegepast, maar is Evans de eerste die het in een boek structureert en systematiseert.

### 3.5.1. Het rijke domein

Bij Domain Driven Design ligt de nadruk op een rijk domein. Door juist nadruk te leggen op het domein en de business logica, zou een project dynamischer, onderhoudbaarder en vaak ook nog efficiënter worden. Door DDD toe te passen wordt een ‘dun’ domein model vermeden, er bestaat een duidelijke splitsing tussen wat belangrijk voor het domein is, en wat niet, zoals bijvoorbeeld een infrastructuur. Deze infrastructurele zaken zouden volgens DDD buiten het domein moeten staan. Het is bijvoorbeeld voor het domein niet belangrijk of de data uit de UI via: XML, JSON, Textfiles etc. binnenkomt dat is geen onderdeel van het domein, het domein is alleen geïnteresseerd in de data zelf. Daarentegen is wel belangrijk dat er op een rekening gestort kan worden via een methode *‘Rekening.stort( Geld )’* bijvoorbeeld.

### 3.5.2. De ‘Ubiquitous language’

Een van de belangrijkste aspecten van DDD is de ‘Ubiquitous language’, dit is de ‘taal’ die gebruikt moet worden om over het domein te communiceren, deze taal moet zowel

door de programmeurs als de domein experts gebruikt worden. Het is van belang duidelijk te krijgen welke termen juist zijn in het specifieke domein. Ook moet er gestreefd moeten worden deze taal in de software te hanteren. Eric Evans geeft hier een voorbeeld van in zijn boek, hij bootst in het boek een gesprek na tussen een ontwikkelaar en een gebruiker.

> "If we give the *Routing Service* an origin, destination, and arrival time, it can look up the stops the cargo will have to make and, well... stick them in the database.(vague and technical)"[4]

In deze quote is te zien, hoe er gebruik wordt gemaakt van technische en vage taal. Er wordt gesproken over een database, iets wat in deze context niet van belang is. Hierdoor ontstaat een grote kans dat de gebruiker het niet begrijpt en dat vervolgens langs elkaar heen wordt gepraat. Het volgende voorbeeld is een duidelijkere versie van het vorige citaat:

> "A *Routing Service* finds an *Itinerary* that satisfies a *Route Specification*. (concise)"[4]

In deze quote is te zien dat er duidelijkere begrippen zijn ontstaan, hierdoor kan er duidelijker tussen de ontwikkelaar en de domein expert gecommuniceerd worden. Routing Service, Itinerary en Route Specification zijn allen duidelijk termen met een betekenis.

Deze taal moet ook terug komen in het model, bijv. door een klasse te definiëren genaamd: *Routing Service* die via een *findItinerary( Route Specification )* methode, een *Itinerary* teruggeeft. Zodoende komt de 'Ubiquitious language' terug in de code. 'Ubiquitious' betekent dan ook 'alomaanwezig', de taal moet overal terugkomen.

Het is belangrijk dat alles in het domein beschrijvend en specifiek is. Stel er bestaat een object Klant en deze kan verhuist worden, dan zal er ook een functie 'verhuis( Adres )' moeten zijn i.p.v. 'setLocatie(String, Int)', met methodes zoals 'verhuis' is duidelijk wat het doel is: de klant verhuizen. Een methode zoals 'setLocatie' daarentegen is onduidelijk, het kan aangeroepen worden tijdens een verhuizing, het instantiëren van het object of misschien wordt het wel gebruikt om de huidige locatie van een klant a.d.h.v. GPS data aan te passen.

### 3.5.3. Model Driven Design

Bij Domain Driven Design is het model het belangrijkst. Het domein dat in de werkelijkheid bestaat moet vertaald worden naar een model in software. Dit kan een objectenmodel zijn, maar dat hoeft niet, zo zou het gedrag van een dier eventueel beter te modelleren zijn in Prolog.

Het is belangrijk dat er vanuit het model gebouwd wordt. Als de code veranderd dan veranderd in principe het model mee. Het model moet als basis dienen voor de code. Het 'juiste' model ontstaat ook niet in een keer, daarvoor zijn meerder iteraties nodig. Daarom adviseert Evans om geen gebruik te maken van gescheiden analyse en design ontwerpen, het juiste model ontstaat volgens Evans tijdens de ontwikkeling ervan. Zowel het technische gedeelte van het model als de semantische betekenis ervan, komt in de

ontwikkelfase aan bod. Evans adviseert dat iedereen die aan het model werkt, ook de bijbehorende code moet aanpassen, geen grote analyses en design fase's vooraf, het model wordt gevormd door met deze te werken.

Het gewenste resultaat is een model waar zowel de belangrijke concepten van het domein naar voren komen, als een technisch degelijke implementatie. Dit kan volgens Evans alleen als er een iteratief proces is waar constant gekeken wordt of het model wel 'klopt' met het domein en of de technische implementatie niet 'clumsy' is:

> "Demand a single model that serves both purposes well, in addition to supporting a robust ubiquitious language."[4]

In het uiteindelijke model moet ook de 'ubiquitious language' terug komen, deze ondersteund het model met duidelijke termen en geeft daardoor het model een concrete vorm.

### 3.5.4. Implementatie

Het laatste wat van belang is om te weten is hoe het dit model dat geïmplementeerd moet worden. Evans heeft hier een aantal technieken en patterns voor die in het tweede deel van het boek worden beschreven. Echter is iteratief werken en refactoring ook van belang, dit onderwerp wordt in het derde deel van het boek vooral behandeld.

Deze aspecten zijn de wat praktischere delen van het boek. Hier zal ik dan ook niet al te diep op ingaan, het is genoeg om te weten dat met deze delen een brug wordt geslagen tussen de concepten (Model Driven Design, Ubiquitious language) en de implementatie ervan. In 7.2.2 komen een aantal van deze technieken in de praktijk aan bod.

### 3.5.5. Samenvatting

Domain Driven Design is een methodiek om een rijk domein te bouwen. Door met het model een domein op een juiste wijze te weerspiegelen onstaat er een onderhoudbare en duidelijkere applicatie. Dit doel is te bereiken door het model centraal te stellen en te communiceren via de 'ubiquitious language'. Een iteratieve werkwijze past het beste bij Domain Driven Design.

Sogyo maakt al enkele jaren actief gebruik van deze methodiek, daarom was het van belang deze methodiek mee te nemen in het onderzoek. Zodat er kan worden aangetoond of een Rule engine in een de bestaande ontwikkkelmethodiek past.

## 3.6. Opgeleverde producten

De producten zijn in principe in twee fase's opgeleverd: de onderzoek en implementatiefase.

### 3.6.1. Onderzoek

In de onderzoeksfase werd een enkel eindproduct opgeleverd en een aantal tussenproducten als milestones. Zodra ik bij een van deze milestones aankwam is het tussenproduct besproken met mijn bedrijfsbegeleider.

Hier volgt een korte lijst met de uiteindelijke producten:

- Het Plan van Aanpak.
- Eerste shortlist.
- Tweede shortlist.
- Uiteindelijke uitgekozen product.

Over deze producten wordt meer gesproken in hoofdstuk 6.De bovenstaande producten zijn vooral voor Sogyo van belang, het leerproces is daarentegen wel weer relevant voor de scriptie dus ook voor de Hogeschool van Utrecht.

### 3.6.2. Implementatie

In de implementatiefase zijn de volgende producten van belang:

- Het Proof of Concept.
- De scriptie.

Het Proof of Concept moet aantonen of een Rule engine geschikt is voor gebruik binnen Sogyo. Dit product is dus voor Sogyo van groot belang. De scriptie is voor beide partijen van belang, aangezien er een aantal conclusies in staan die relevant zijn voor Sogyo. Op het Proof of Concept wordt in hoofdstuk 7 verder op teruggekomen.

# Deel II.

# Opdrachtanalyse

# 4. Inleiding Rule engines

Dit hoofdstuk dient als een korte inleiding op het onderwerp Rule engines. Dit hoofdstuk verschaft informatie over wat een Rule engine is en waarvoor het doelt.

## 4.1. Concepten Rule engines

*Een Rule engine is een systeem dat regels gebruikt om een aantal acties te ondernemen.* Zo is de definitie (kort door de bocht) in één zin te omvatten. Waarschijnlijk is het nog niet duidelijk wat met deze definitie daadwerkelijk wordt bedoelt. Het is daarom belangrijk om een aantal concepten eerst te begrijpen, zodat de definitie duidelijker wordt.

### 4.1.1. Wat is een regel/rule?

Wat is een regel ofwel rule nou eigenlijk? In het dagelijks leven hebben we ook vele regels waar wij ons aan moeten houden. Wat wij hiermee eigenlijk doen is restricties opstellen. Dit is simpel gezegd hetzelfde wat bij een Rule engine ook wordt gedaan.

Zo zou je bijvoorbeeld kunnen stellen dat mensen die te hard rijden een boete moeten krijgen. Dit wordt dan een regel: 'Mensen die te hard rijden krijgen een boete'. De vraag komt al snel naar boven wat 'te hard rijden' in de context van de regel is. Als antwoord op deze vraag wordt er een restrictie opgelegd namelijk: 'alles boven de 70 km per uur'. Of dit nou eerlijk is of niet, duidelijk is het in ieder geval wel. Iedereen die boven de 70 km per uur rijdt krijgt een boete, iedereen die dit niet doet krijg geen boete. Door deze restrictie wordt duidelijk wat 'te hard rijden' is.

### 4.1.2. Wat is een feit/fact?

In de vorige paragraaf werd al gesproken over feiten. In een Rule engine zijn feiten niet erg anders dan in het echte leven. De regels in een Rule engine maken beslissingen aan de hand van feiten. 'Jan heeft bruin haar en is 30 jaar oud', dat is een feit. Aan de hand dit soort feiten maakt de Rule engine beslissingen.

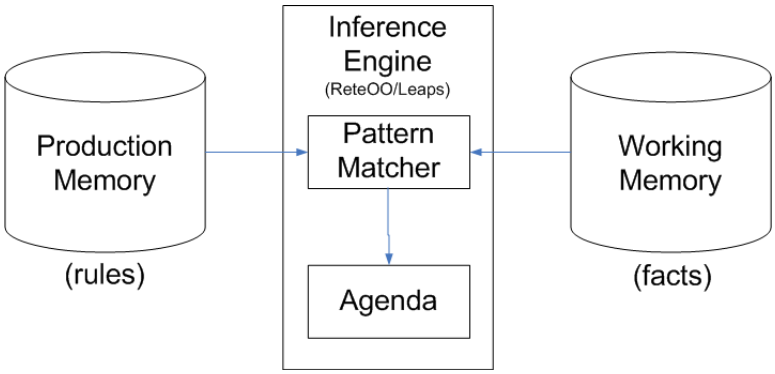
Feiten kunnen net zoals in het echte leven ook veranderen; 20 jaar later heeft Jan namelijk grijs haar en is er geen enkele bruine haar meer te bekennen. Dat 'Jan grijs haar heeft' is nu een feit. De consequentie is dat een Rule engine vervolgens andere beslissingen zal moeten maken.

## 4.2. Wat is een Rule engine?

Een Rule engine is niets meer dan een systeem dat kijkt of één of meerdere feiten aan opgestelde regels voldoen. Hierbij komt uiteraard de nodige complexiteit kijken zoals de volgorde van de uitvoer van de regels. Ondanks deze complexiteit blijft de kernfunctionaliteit van een Rule engine hetzelfde.

Het idee is dat als de kennis, in de vorm van regels, samen met de feiten in het systeem zitten. De Rule engine zelf kan beredeneren wat gedaan moet worden. Als je ooit hebt gewerkt met de programmeertaal Prolog klinkt dit wellicht bekend in de oren. De filosofie van Prolog (het werkt niet geheel zo in de praktijk), is dat er op een logische en declaratieve manier het probleem beschreven wordt en dat Prolog zelf uitzoekt, wat het antwoord op het probleem is. In plaats van dat het vinden van de oplossing in procedurele stappen wordt beschreven.

Wat er wordt gedaan bij een Rule engine is dat er een 'knowledge base' wordt opgesteld. Deze 'knowledge base' bevat menselijke kennis in de vorm van regels. Deze regels worden in het geheugen gehouden en de feiten worden hierop getoets. Het toetsen wordt meestal gedaan door een inference engine.




Figuur 4.1.: Delen van een Rule engine

In figuur 4.1 is te zien hoe de delen van een Rule engine in elkaar zitten. Zo worden de regels en feiten apart bijgehouden en is te zien dat deze beide als input worden gebruikt voor de inference engine. Over hoe de inference engine werkt wordt in de volgende secties in detail op ingegaan.

### 4.2.1. Inference engine

De meeste Rule engines maken gebruik van een inference engine. De inference engine is het brein van een Rule engine. De inference engine maakt beslissingen door feiten te 'matchen' met regels. Dit matchen heet 'pattern matching'. Dit proces is vergelijkbaar met het 'toetsen' van een woord of zin aan een reguliere expressie. Alleen gebeurt hier het met feiten en regels. Een 'pattern match taal' kan kan gezien worden als een reguliere expressie voor regels en feiten.

De inference engine trekt een aantal conclusies. Zoals 'Jan rijdt inderdaad harder dan 70 km per uur', dit is een 'match'. Vervolgens wordt ervoor gezorgd, dat er eventueel een aantal acties worden uitgevoerd. In dit geval dat Jan een boete krijgt.

### 4.2.2. Rule engine zonder inferencing

In de vorige paragraaf is gesproken over een inference engine in een Rule engine. Van oorsprong was de inference engine een onderdeel van elke Rule engine. Tegenwoordig zijn er Rule engines die de regels compileren naar code en niet meer gebruik maken van een inference engine. Dit betekent dat een deel intelligentie weg valt, een inference engine is namelijk het deel van de Rule engine wat redeneert. Het weg laten van de inference engine kan zorgen voor een snelheidswinst, maar als gevolg: het verlies van functionaliteit. Zie 5.5 voor meer informatie over inference engines en de consequenties van het weglaten van een inference engine.

### 4.2.3. Regels in een Rule engine

Hoe regels precies gedefinieerd worden in een Rule engine verschilt per product, ze hanteren echter wel dezelfde vorm.

- LHS. Left Hand Side. De LHS is de 'als' ofwel het predicaat.
- RHS. Right Hand Side. De RHS is de 'dan' ofwel de conclusie.

Dit is te vergelijken met hoe wij normaal ook regels definieren:

```
LHS -> [Als Een persoon harder dan 70 km per uur rijdt]
RHS -> [Dan Krijgt hij een boete.]
```

Waar 'Als' het 'LHS' gedeelte is en 'Dan' het 'RHS' gedeelte. Dit is in principe wat nagebootst wordt in een Rule engine. Als deze regel verder ontleed wordt zien we welk patroon 'gematched' moet worden: alle personen in het systeem die op dit moment harder rijden dan 70 km per uur.

Als een feit voldoet aan een regel: Jan rijdt harder dan 70. Wordt vervolgens de regel geactiveerd en wordt het 'dan'/'RHS' gedeelte uitgevoerd.

### 4.2.4. Feiten in een Rule engine

Feiten worden net zoals regels in het systeem gezet. Hoe dit gebeurd en of feiten later kunnen worden toegevoegd/aangepast verschilt per implementatie. Met bijna alle Java Rule engines kan gebruik worden gemaakt van POJO's (Plain Old Java Objects) als feiten. Deze POJO's kunnen als het ware geïnjecteerd worden in het systeem, dit is vanuit Java code mogelijk.

### 4.2.5. Activatie en deactivatie van rules/regels

Het is dus duidelijk dat een Rule engine regels en feiten bevat en dat er regels geactiveerd kunnen worden. Maar wat is activatie nou werkelijk? Als een feit voldoet aan de regel, vindt er een activatie plaats. Dit betekent dat een Rule engine herkent dat een regel uitgevoerd moet gaan worden. Als er iets in het systeem veranderd waardoor de regel niet meer geld, dan wordt de regel weer gedeactiveerd.

## 4.3. Voordelen

De voordelen van een Rule engine t.o.v. een traditioneel objectenmodel:

1. Regels zijn gescheiden van programma-logica.
2. Regels kunnen declaratief en natuurlijker worden opgesteld. Regels beschrijven wat er moet gebeuren, niet hoe.
3. Rule engines kunnen hele complexe problemen oplossen.
4. Kennis is gecentraliseerd. Het zit in de engine en niet verspreid over verschillende objecten.

## 4.4. Nadelen

De nadelen van een Rule engine t.o.v. een traditioneel objectenmodel:

1. Rule engines zijn redelijk complex.
2. De regels zitten in een apart systeem en niet meer in het domein (denk aan DDD).
3. Rule engines zijn langzamer dan een algoritmische oplossing.
4. Ontwikkelaars moeten een nieuwe tool leren gebruiken.

## 4.5. One size fits all?

Rule engines zijn, net zoals alle tools, niet *de oplossing* voor elke situatie. Al snel kan worden geconstateerd dat een Rule engine slechts in hele specifieke situaties gebruikt kan worden. In de 'Drools manual' staat een goede opsomming van eisen voor het probleem, wil er gebruik worden gemaakt van een Rule engine.

> • "The problem is just too fiddle for traditional code. The problem may not be complex, but you can't see a non-fragile way of building a solution for it.

- The problem is beyond any obvious algorithmic solution. It is a complex problem to solve, there are no obvious traditional solutions, or basically the problem isn't fully understood.
- The logic changes often. The logic itself may even be simple but the rules change quite often. In many organizations software releases are few and far between and pluggable rules can help provide the 'agility' that is needed and expected in a reasonably safe way.
- Domain experts (or business analysts) are readily available, but are nontechnical. Domain experts often possess a wealth of knowledge about business rules and processes. They typically are nontechnical, but can be very logical. Rules can allow them to express the logic in their own terms. Of course, they still have to think critically and be capable of logical thinking. Many people in nontechnical positions do not have training in formal logic, so be careful and work with them, as by codifying business knowledge in rules, you will often expose holes in the way the business rules and processes are currently understood."

[15]

Rule engines worden tegenwoordig veel gebruikt in applicaties waar het domein complex is en veel business rules zijn. Denk hierbij aan assemblage systemen, verzekering systemen en medische systemen.

Hiervoor zijn Rule engines geschikt. Voor systemen waar de business rules om de haverklap veranderen is het wellicht ook interessant. Voor een klein domein met een matige complexiteit is een Rule engine minder geschikt. Het zorgt dan voor onnodige complexiteit.

De conclusie is dat de keuze voor het gebruik van een Rule engine, goed overwegen moet worden. Het gebruik van een Rule engine is alleen te verantwoorden in een aantal situaties.

# 5. Gebruik van een Rule engine

In dit hoofdstuk zal een korte inleiding worden gegeven over het gebruik van Rule engines en de bijbehorende technieken die gebruikt worden.

## 5.1. Rete algoritme

Het Rete algoritme is voor het eerst beschreven in: *Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem.* In verloop der tijd is het nog een aantal keer door verschillende producten efficiënter gemaakt, de oplossing blijft echter grotendeels hetzelfde.

Wat het algoritme beschrijft is op een slimme wijze 'pattern matching' uit te voeren. Deze techniek ligt ten grondslag aan veel Rule engines van vandaag. De technieken die in de hoofdstuk worden beschreven zijn dan ook Rete technieken. Conflict resolution, activatie en uitvoer van regels; zouden met een ander algoritme waarschijnlijk anders opgelost worden

## 5.2. Knowledge Base

Het eerste wat gedaan moet worden is het opbouwen van de knowledge base, deze wordt vaak afgekort naar KB. Dit wordt gedaan door regels te definiëren in een bestand, die bij de initialisatie van het systeem wordt ingelezen. De feiten worden later aan het systeem toegevoegd. Een aantal Rule engines geven de mogelijkheid om de knowledge base tijdens run-time aan te passen. De knowledge base is vanaf het moment van inladen de 'kennis' die de Rule bezit, aan de hand van deze 'kennis' kan de inference engine gaan beredeneren.

## 5.3. Activatie van regels

Een Rule engine reageert op verandering in het systeem. Een Rule engine houdt de feiten in het systeem in de gaten en probeert, alleen als feiten gewijzigd zijn of ingevoerd, deze te 'matchen'. Zodra er een match is gevonden, wordt de regel geactiveerd en op de agenda gezet. Een activatie zorgt er dus voor dat regels op de agenda worden gezet maar niet uitgevoerd, later worden de regels via een functie sequentieel worden uitgevoerd.

Een agenda houdt bij in welke volgorde de regels uitgevoerd moeten worden. De regels worden normaliter op volgorde uitgevoerd. Alleen ontstaat er soms een probleem, wat gebeurt er als twee regels tegelijk geactiveerd moeten worden? Stel er zijn twee regels: A en B, A en B hebben beide dezelfde activatie voorwaarden. Zodra A geactiveerd wordt

moet B dus ook geactiveerd worden, welke regel moet eerst? Wat er nu ontstaat is een conflict. Om dit op te lossen maken Rule engines gebruik van Conflict resolution.

## 5.4. Conflict resolution

Conflict resolution betekent: het oplossen van conflicten. Hoe een Rule engine dit doet verschilt per implementatie. De meeste Rule engines kennen het concept: 'salience'. De hoogte van de 'salience' bepaalt hoeveel prioriteit een regel krijgt. Een regel met een 'salience' van 10 heeft hierdoor voorrang op een regel met een 'salience' van 5.

De meeste Rule engines kennen ook agenda groepen, met agenda groepen kan er een bepaalde flow worden gecreëerd. Alleen bij de agenda groep die focus heeft worden regels uitgevoerd. Regels kunnen ten alle tijden worden geactiveerd krijgt de agenda groep van regel focus, dan worden de desbetreffende regels uitgevoerd.

Voor een rule engine resulteert de simultane activatie van twee of meerdere regels altijd in conflict. Het hoeft echter voor de gebruiker niets uit te maken, in welke volgorde de conflicterende regels worden uitgevoerd. Wordt er gekozen om geen gebruik te maken van conflict resolution dan is de executie van de desbetreffende regels arbitrair.

## 5.5. Inference engine

Tegenwoordig zijn er een aantal producten op de markt die geen gebruik van een inference engine en het Rete algoritme. De voordelen van deze methode zijn:

- Het resulteert in betere performance.
- Er is minder theoretische kennis nodig.
- De regels zijn vaak minder complex.
- De Rule engine implemantatie is simpeler.
- Er is minder snel een DSL nodig voor specificeren van regels.

Er zijn ook een aantal nadelen:

- Een deel van de intelligentie valt weg.
- Alles moet handmatig gespecificeerd worden, geen conflict resolution meer etc.
- Bij grote aantallen regels kan het onoverzichtelijk worden.
- Er zijn minder complexe structuren mogelijk.

De inference engine is in principe de brein van de applicatie. De inference engine redeneert over zaken aan de hand van regels en feiten. Als de inference engine weg valt valt ook de mogelijkheid om te redeneren weg.

### 5.5.1. Visual Rules

Visual Rules is een product wat getest is voor het onderzoek. Visual Rules maakt geen gebruik van een inference engine, de regels moeten met behulp van een 'rule-flow' worden

beschreven. Een 'rule-flow' is te vergelijken met een flow-chart, op deze wijze moest het beslissingsproces opgezet worden. Pattern matching is wel een onderdeel van Visual Rules engine, maar de gebruiker moet zelf definiëren in welke volgorde de regels uitgevoerd worden.

De activatie en deactivatie van regels gaat ook anders. Als een regel geactiveerd wordt, wordt de regel ook direct uitgevoerd. Een agenda bestaat dus niet. Als er gedefinieert is dat regel A niet meer moet gelden als B geld, dan is dit in een Rule engine met een inference engine geen probleem. Regels worden eerst geactiveerd en kunnen ook weer gedeactiveerd worden. Zonder inference engine is dit moeilijker op te zetten A wordt namelijk direct uitgevoerd, er bestaat geen activatie.

Een Rule engine zonder inference engine is altijd stateless. Feiten worden niet bijgehouden door het systeem, er wordt een enkele keer overheen gelopen. Elke keer dat er feiten gecontroleerd moeten worden, wordt er een nieuwe sessie aangemaakt.

Door dit gedrag wordt de Rule engine procedureler. Een aantal geavanceerde functies van traditionele Rule engines zijn nu ook niet meer mogelijk, zoals het bijhouden van de waarheid.

### 5.5.2. Wanneer is inferencing noodzakelijk?

De meeste problemen kunnen echter wel sequentieel worden opgelost, ik vroeg mij dus af wat concreet een voorbeeld is waar inferencing noodzakelijk is. Via de Drools mailing list kreeg ik het volgende voorbeeld:

> "Wolfgang, you've basically described the system I've been working on for two years, the Early Alerts System at Southwest Airlines. So yes, Tim, there are real world examples. :) In my case EAS monitors the state of our aircraft fleet (and soon the state of customer bookings) and alerts dispatchers to problems. The actions of the dispatchers and ground operations personnel feeds back into the system and changes the aircraft state.
>
> The app was actually designed in a semi stateless manner at first, but it quickly became apparent that the state of all aircraft had to be tracked to reduce rule complexity and increase performance. Otherwise, every time a significant operation had to be performed, the current state of the aircraft had to be retrieved from external system, which did not scale well at all. Also the state being stored locally means you can continually update it and write rules that fire off the state of groups of aircraft, which would be even more untenable if handled in a stateless manner. (i.e. when monitoring all aircraft at an airport we'd have to periodically load the state of the entire airport and run rules, when in statefull mode all we have to do is subscribe to the aircraft events, a simpler and less resource intensive process.)"[1]

# 6. Onderzoek

## 6.1. Introductie onderzoek

Allereerst een klein overzicht met de punten die in dit hoofdstuk behandelt worden:

1. Als eerste wordt de onderzoeksvraag herhaald zoals deze is opgezet in het Plan van Aanpak.
2. Vervolgens wordt er een overzicht gegeven hoe het onderzoek is aangepakt en in welke delen het is opgedeeld.
3. Hierna volgt er een beschrijving van de verschillende fases.
4. Ten slotte wordt er gekeken of en hoe de onderzoeksvraag beantwoord is.

Voor een gedetailleerder overzicht van de planning en taken, zal gerefereerd moeten worden naar het Plan van Aanpak.

### 6.1.1. Onderzoeksvraag

Tijdens het opstellen van mijn Plan van aanpak is een onderzoeksvraag geformuleerd. Deze luidt als volgt:
"Wat is de meest geschikte Rule engine voor Sogyo?".
Deze vraag is op te delen in de volgende deelvragen:

- Welke Rule engines zijn er?
- Wanneer en waarom is een Rule engine geschikt?
- Wat voor consequenties heeft een Rule engine op Domain Driven Design?
- Zijn er Rule engines die rekening houden met Domain Driven Design?
- Welke eisen stelt Sogyo aan een Rule engine?
- Zijn er in voorgaande projecten elementen geweest die vervangen zouden kunnen worden door een Rule engine? Zo ja welke?

Deze vraag heb is getracht te beantwoorden in het onderzoek. In meerdere fases is gewerkt aan het beantwoorden van de onderzoeksvraag en deelvragen. De vragen zijn beantwoord d.m.v. de verschillende taken: een vooronderzoek, verschillende interviews, een inventarisatie en ten slotte het testen van de verschillende producten.

### 6.1.2. Fase's

Er zijn meerdere fase's gespecificeerd in het Plan van Aanpak:

1. Plan van Aanpak schrijven.

2. Vooronderzoek.
3. Inventarisatie (hier kwam een eerste shortlist uit voort).
4. Producten uit de eerste shortlist aan technische tests onderwerpen.
5. Criteria vanuit Sogyo opstellen.
6. Opnieuw beoordelen van producten a.d.h.v. eisen Sogyo.
7. Resultaten verwerken scriptie.

Gedurende het onderzoek, heb ik gemerkt dat verschillende fase's door elkaar gingen lopen. Zo liep het vooronderzoek en de inventarisatie gedeeltelijk gelijk. En tijdens het testen van de producten, moest er regelmatig nog onderzoek gedaan worden naar de theorie. In elke fase was er wel een moment waar nog wat taken moest uitgewerkt moesten worden uit de vorige fase.

De criteria vanuit Sogyo opstellen is a.d.h.v. interviews gedaan, deze interviews hebben geholpen met de beeldvorming tijdens de inventarisatie en test fases. Het houden van interviews is dus uiteindelijk geen aparte fase geworden. Zodoende kon na het opstellen van een tweede shortlist, eigenlijk al direct advies gegeven worden aan Sogyo. De eisen waren reeds geïnventariseerd en duidelijk.

De volgende fases, zullen in detail behandelt worden:

1. Vooronderzoek.
2. Inventarisatie, eerste shortlist.
3. Technische evaluatie, tweede shortlist.
4. Resultaten verwerken in scriptie, onderzoeksvraag beantwoorden.

De taken uit de overige fases zijn wel uitgevoerd, maar zullen besproken worden in de hierboven gespecificeerde fases.

### 6.1.3. Resultaat

Het opgeleverde resultaat is een adviesproduct, met als onderdeel informatie over Rule engines wat in hoofdstukken 4 en 5 terug te vinden is. De combinatie van deze informatie en de productevaluatie leidt tot een adviesproduct. Hier wordt op terug gekomen aan het einde van dit hoofdstuk.

## 6.2. Vooronderzoek

### 6.2.1. Doel

Het doel van het onderzoek was om de volgende deelvragen te beantwoorden:

- Welke Rule engines zijn er?
- Wanneer en waarom is een Rule engine geschikt?
- Wat voor consequenties heeft een Rule engine op Domain Driven Design?
- Zijn er Rule engines die rekening houden met Domain Driven Design?
- Welke eisen stelt Sogyo aan een Rule engine?

- Zijn er in voorgaande projecten elementen geweest die vervangen zouden kunnen worden door een Rule engine? Zo ja welke?

Dit heb is gedaan d.m.v. verschillende activiteiten, deze activiteiten worden hieronder beschreven.

### 6.2.2. Onderzoek naar de theorie

Dit deel van het onderzoek is uitgevoerd door het lezen van verschillende resources. Zowel op internet als in de handleiding van verschillende producten, is redelijk wat te vinden over Rule engines en de theorie erachter. De taal Prolog is ook een aantal dagen bestudeerd, aangezien dit een soort Rule engine is. Prolog geeft inzicht in verschillende concepten zoals bijvoorbeeld; forward en backward chaining.

Vooral de algemene vragen zoals: 'Welke Rule engines zijn er?'. Zijn door een dergelijk onderzoek goed te beantwoorden. De vragen over DDD in combinatie met een Rule engine zijn op deze wijze echter wat moeilijker te beantwoorden. Om deze vragen te beantwoorden is o.a. uit het Domain Driven Design boek *"Domain-Driven Design: Tackling Complexity in the Heart of Software"* van Eric Evans gelezen, en een interview gehouden met een medewerker van Sogyo, die veel kennis heeft over Domain Driven Design.

### 6.2.3. Interviews

Er zijn interviews gehouden met verschillend Sogyo medewerkers. Korte beschrijvingen van deze interviews zijn te vinden in bijlage A. Ook zijn regelmatig informele gesprekken gehouden met medewerkers over het onderwerp. Wat met deze interviews bereikt moet worden was het volgende:

- De vragen over DDD beantwoorden.
- Welke eisen stelt Sogyo aan een Rule engine.
- Alvast een globaal idee krijgen van geschikte projecten.

### 6.2.4. Evaluatie

Het vooronderzoek liep voorspoedig. De vragen konden goed beantwoord worden. Bij de inventarisatie kwam ik erachter dat het handig was om al enig idee te hebben wat de eisen vanuit Sogyo zijn. Daarom ben ik al in deze fase met de interviews begonnen. Met deze kennis kon gerichter gezocht worden, dit had ik bij het opstellen van het Plan van Aanpak niet voorzien, maar blijkt achteraf de juiste keuze te zijn geweest.

Wat ik had kunnen verbeteren, is de informatie die ik verzamelde beter structureren. Nu moest ik achteraf nog verschillende bronnen bekijken tijdens het schrijven van de inleiding over Rule engines. Had ik dit beter had bijgehouden (stukken op slaan die intressant zijn), dan zou dit mij tijd hebben gescheeld.

## 6.3. Eerste inventarisatie

### 6.3.1. Doel

Het doel van de eerste inventarisatie, is een overzicht te krijgen van de verschillende producten op de markt.

### 6.3.2. Methodiek

De gebruikte methodiek voor de inventarisatie was eenvoudig. Op internet is veel te vinden over de verschillende producten, dit deel van het onderzoek is uitgevoerd met behulp van: Google, fora en blogs. Via deze media is een beeld gevormd over de huidige producten op de markt.

In deze fase was informatie nodig uit mijn vooronderzoek, om de verschillende aspecten van de producten te kunnen vergelijken. Tevens komt veel theoretische informatie uit de handleidingen van de verschillende producten, zodoende werd in het vooronderzoek al geïnventariseerd.

### 6.3.3. Overzicht eerste inventarisatie

In tabel 6.1 staat een overzicht van de eerste inventarisatie. Aan de verschillende criteria zijn waardes toegekend. Allereerst is een iventarisatie uitegevoerd van de huidige producten op de markt. Vervolgens zijn er waardes toegekend aan deze producten, dit wordt in 6.4 beschreven. Door een inventarisatie uit te voeren is eerst een globaal beeld gevormd, waardoor een vergelijking van producten makkelijker wordt.

| | Rule engine | Documentatie | Licentie | Volwassenheid | Features | Activiteit | Support | Relevantie |
|---|---|---|---|---|---|---|---|---|
| 1 | Jess | ++ | Non-commercial | ++ | + | – | X | ++ |
| 2 | Drools | ++ | ASL | ++ | ++ | ++ | O | ++ |
| 3 | Prova | + | GPL | + | – | ++ | X | - |
| 4 | Take | + | ASL | – | + | – | X | + |
| 5 | OpenLexicon | – – | Mozilla | X | X | – | X | – – |
| 6 | OpenRules | + | GPL/ Proprietary | + | + | + | O | + |
| 7 | Zilonis | – | X | – – | – | – – | X | – |
| 8 | Hammurapi Rules | – – | LGPL | + | – | + | X | + |
| 9 | JEOPS | – | LGPL | X | + | – – | X | – – |
| 10 | Termware | + | X | X | + | + | X | – |
| 11 | Algernon | – – | MPL | X | – | – – | X | + |
| 12 | TyRuBa | – | BSD | + | – | – | X | – |
| 13 | Esper | + | GPL/ Proprietary | ++ | + | + | O | - |
| 14 | ILog JRules | + | Proprietary | ++ | ++ | + | O | + |
| 15 | Visual Rules | ++ | Proprietary | ++ | ++ | ++ | O | ++ |

Tabel 6.1.: Eerste inventarisatie van Rule engines, de producten zijn op verschillende criteria beoordeelt.

– – Erg slecht
– Slecht
+ Redelijk tot Goed
++ Erg goed
**O** Aanwezig
**X** Geen informatie

## 6.4. Eerste shortlist

Na de eerste inventarisatie, was de tweede fase van het onderzoek om de eerste shortlist op te stellen. Dit houdt in dat de verschillende producten uit de eerste inventarisatie, op verscheidene aspecten werden onderzocht. Bij het opstellen van de eerste shortlist, is voornamelijk gekeken naar de algemene aspecten van de geïnventariseerde producten.

Uit deze fase van het onderzoek kwam een lijst met een viertal producten die getest zijn aan de hand van technische criteria.

### 6.4.1. Doel

Het doel van de eerste shortlist is om meer informatie vast te leggen dan alleen de naam en een korte omschrijving, er wordt hier naar algemene aspecten gekeken die op het eerste gezicht te beoordelen zijn.

Met een scoring, aan de hand van algemene criteria, kan beoordeelt worden, welke producten degelijk zijn. Als een product niet van voldoende kwaliteit is (daardoor niet degelijk), heeft het weinig zin om deze aan een technische beoordeling te onderwerpen. Een technische beoordeling kost namelijk veel tijd, aangezien dan in meer detail met het product moet worden gewerkt.

### 6.4.2. Selectie en scoring

#### Selectiemethode

Aan de hand van een 'eerste indruk' zijn een aantal criteria opgesteld. Wat hiermee bereikt wilde worden is de producten eruit te filteren, die op algemene punten niet degelijk zijn. De score is echter niet het enige wat bepaalt of een product in de eerste shortlist komt. Als gebruik wordt gemaakt van een ander soort paradigma, is het eventueel toch interessant om het product mee te nemen. Tevens is geprobeerd de beoordeling indien nodig te nuanceren. Stel een product is niet erg volwassen maar wel actief in ontwikkeling, dan is het product wellicht nu niet interessant maar over een aantal maanden wel.

#### Scoring

Na het opstellen van de lijst en het beoordelen van de verschillende punten, is nog een uiteindelijke score gegeven aan de verschillende Rule engines. Dit geeft wat concreter beeld over hoe een Rule engine in het algemeen scoort. De scoring is als volgt:

−− -2
− -1
+ 1
++ 2

Aan de verschillende criteria wordt een gewicht gegeven. Een punt kan 1x, 2x of 3x zo zwaar wegen. Dus een '+' waardering wordt met een gewicht van 2: 2 punten en een '−−' waardering wordt met een gewicht van 2: -4 punten.

Uiteindelijk worden de punten bij elkaar opgeteld. Deze opsomming is de uiteindelijke score, een negatieve score is mogelijk.

### 6.4.3. Toelichting Algemene criteria

In deze sectie volgt een lijst met algemene criteria, inclusief uitleg en de verschillende gewichten:

**Documentatie** Documentatie is een heel belangrijk punt. Documentatie bepaalt namelijk hoe snel ontwikkelaars met het product aan de slag kunnen. Omdat veel producten in de lijst open-source zijn, wordt dit punt zelfs nog belangrijker. Het komt namelijk vaak voor, dat door gebrek aan documentatie, het niet duidelijk is hoe een product gebruikt zou moeten worden. Om het product goed te kunnen testen is het ook erg belangrijk dat er snel aan de slag kan worden gegaan. Zonder documentatie is dit niet mogelijk. Ook is er gekeken of er een beschrijving is van de API, omdat de integratie met Java een belangrijk punt is. Gewicht: **3x**.

**Licentie** Licentie is ook een belangrijk punt. Bij een aantal Rule engines was niet te zien, welke licentie gebruikt werd en een aantal Rule engines gebruikte twee verschillende licenties; voor zowel non-commercieel en commercieel gebruik. Voor de licentie wordt geen score voor gegeven, de licentie bepaalt niet de degelijkheid van het product. Bij het uitkiezen van de producten op de short-list wordt hier echter wel rekening mee gehouden, het is bijvoorbeeld moeilijk om GPL producten te gebruiken. Deze licentie dient namelijk alleen in open-source maatwerk gebruikt worden.

**Volwassenheid** Volwassenheid is van belang omdat dit bepaalt hoe bruikbaar het product is. Het ligt aan het project hoe snel deze volwassen is. Bij open source projecten worden vaker sneller incrementen opgeleverd maar stagneert een product ook sneller. Volwassenheid weegt minder zwaar dan activiteit, software is snel out of date dus als er niet veel ontwikkeld wordt (vooral als de gewenste potentie nog niet bereikt is) is het vaak minder nuttig. Daarom is activiteit belangrijker dan volwassenheid.

De volwassenheid van een product is gemeten door te kijken vanaf wanneer het ontwikkeld werd. De hoeveelheid 'major versions' er opgeleverd zijn en of er literatuur over geschreven is. Als er literatuur over geschreven is, duidt dat meestal aan dat het product tenminste al voor experimentele toepassigen gebruikt kan worden. Gewicht: **1x**.

**Features** Features zijn belangrijk in het opzicht dat ze meer mogelijkheden bieden. Tijdens het opstellen van de shortlist is niet alleen gekeken naar features in de zin van de hoeveelheid extra tools, denk hierbij aan een Eclipse plugin, Web Interface e.t.c. Maar ook of het meerdere paradigma's ondersteund. Hoe kan je regels definiëren e.d. Daarom weegt dit punt vrij zwaar, goed uitgewerkte mogelijkheden duiden meestal op een degelijk product. Om een genuanceerder beeld te geven, is er gekeken naar de features in relatie met de activiteit: is er veel activiteit dan komen de features vanzelf. Gewicht: **2x**

**Activiteit** Activiteit is een ander belangrijk punt. Een aantal producten worden ontwikkeld door grote bedrijven. In het geval van Drools is dit Red Hat, deze producten hebben redelijk wat mankracht en geld tot hun beschikking, bij meeste projecten is dit echter niet het geval. Het is daarom belangrijk om de activiteit te meten. Dit is gemeten door naar de laatst uitgebrachte versie te kijken en wanneer de laatste 'commits' zijn gedaan in het versie beheer pakket. Evenals de activiteit op de fora, indien aanwezig. De activiteit bepaalt hoe lang een product bruikbaar up to date blijft en of eventuele problemen snel opgelost kunnen worden. Gewicht: **2x**

**Support** Een ander criteria is support. Kan er eventuele support geleverd worden voor het product? Support is mogelijk van belang in productie. Ik kan niet beoordelen hoe goed deze support is als deze aanwezig is, aangezien hier vaak betaald voor moet worden. Daarom wordt slecht 1 bonus punt bijgeteld als de mogelijkheid bestaat. Geen gewicht.

**Relevantie** Relevantie betekent in deze context: of een Rule Engine specifiek gemaakt is voor de 'normale' software projecten. Met 'normaal' wordt bedoelt, dat het product niet bedoelt is om bijv. algebraïsche functies op te lossen. De geïnventariseerde eisen van Sogyo werden gebruikt om de relevantie te meten. Gewicht: **2x**

### 6.4.4. Scoring

In tabel 6.2, is een score toegekend aan ieder product.

| Pos. | Rule engine | Score |
|---|---|---|
| 1 | Drools | 21 |
| 2 | Visual Rules | 21 |
| 3 | ILog JRules | 14 |
| 4 | Jess | 12 |
| 5 | OpenRules | 11 |
| 6 | Esper | 10 |
| 7 | Take | 5 |
| 8 | Prova | 4 |
| 9 | Termware | 3 |
| 10 | Hammurapi Rules | -2 |
| 11 | TyRuBa | -8 |
| 12 | JEOPS | -11 |
| 13 | Zilonis | -12 |
| 14 | Algernon | -12 |
| 15 | OpenLexicon | -18 |

Tabel 6.2.: Scoring van producten uit tabel 6.1

### 6.4.5. Productkeuze

In de bijlage B zit een uitgebreid overzicht met de verschillende producten en waarom deze op de eerste shortlist staan. Voornamelijk is gekozen voor producten die; relatief goed zijn doorontwikkeld, niet te duur zijn, een redelijk aantal features bevatten en voldoende documentatie hebben.

### 6.4.6. Uiteindelijke eerste shortlist

In tabel 6.3, staat de uiteindelijke shortlist.

| Pos. | Rule engine |
|---|---|
| 1 | Drools |
| 2 | Visual Rules |
| 3 | Jess |
| 4 | OpenRules |

Tabel 6.3.: Meegenomen producten uit tabel 6.2

### 6.4.7. Evaluatie: Eerste shortlist

Deze fase van het onderzoek verliep voorspoedig. De manier waarop de fase opgezet was bleek in de praktijk goed uit te pakken. Ik heb per product ongeveer even veel tijd vrijgemaakt om de verschillende criteria te toetsen. Gaandeweg kwam ik erachter, dat een gelijk verdeelt gewicht niet eerlijk was. Een product kon redelijk slecht scoren, omdat hij op minder belangrijke punten slecht scoorde. Daarom heb ik besloten om een bepaald gewicht aan de verschillende criteria te hangen.

Opmerkelijk waren de vele producten die onvoldoende uit de test kwamen. Dit komt, in mijn opinie, omdat er veel open-source producten op de lijst staan. Waarvan een aantal niet erg zijn uitontwikkeld, of omdat de ontwikkelingen al zijn gestagneerd. Door dit feit was het het niet moeilijk om de vier verschillende producten uit te kiezen.

Toen ik al bezig was met het opstellen van de eerste shortlist, kwam ik het Visual Rules product tegen. Visual Rules had ik op dat moment nog niet meegenomen met de inventarisatie, alleen was wel interessant. Daarom heb ik Visual Rules alsnog een score toegekend en is uiteindelijk zelfs op de eerste shortlist gekomen.

De tijd die ik nodig had voor de eerste shortlist was goed ingecalculeerd en ik was dan ook voor de deadline nog klaar.

Voor de producten op de eerste shortlist heb ik een verlengde trial licentie kunnen bemachtigen. Zo konden de producten gedurende de gehele stage getest worden.

## 6.5. Tweede shortlist

### 6.5.1. Methodiek

In tabel 6.3 staat een lijst met producten die op technische criteria beoordeeld zijn. De producten zijn beoordeeld op meerdere criteria, die zijn gestest door middel van een testproject. Dit testproject was Conway's game of life implementatie.

#### Doel

Mijn doel was een gedegen technische beoordeling te geven van de producten op de eerste shortlist. Ik wilde meer kennis maken met de verschillende Rule engines en zo de 'ins' en 'outs' van de producten leren kennen. Zodat met deze kennis een beter advies aan Sogyo kon worden gegeven over de geschiktheid van de producten.

#### Conway's game of life

Conway's game of life is interessant voor Rule engines omdat het een simulatie is, gebaseerd op een aantal regels. Een game of life bord bestaat uit een matrix van cellen, deze matrix is van willekeurige grote. Binnen deze matrix sterven of leven cellen aan de hand van een aantal regels, deze regels zijn als volgt:

1. Een levende cel met meer dan 3 buren gaat dood door overbevolking.
2. Een levende cel met minder dan 2 buren gaat dood door eenzaamheid.
3. Een dode cel komt tot leven als deze precies drie buren heeft.

Dit zijn een klein aantal regels die weliswaar de kern zijn van het programma. Ik heb de Game of Life daarom uitgekozen als test project. Het project leent zich goed om in een Rule engine gemaakt te worden.

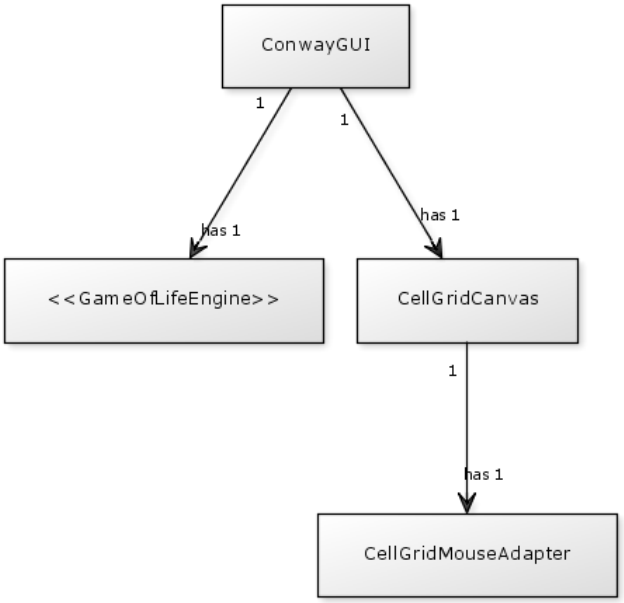
#### Aanpak

Deze fase is als volgt aangepakt:

1. Eerst heb ik de 'GUI','CellGrid' en andere benodigde code geschreven in Java.
2. Herhaal voor elke Rule engine:
   a) Implementeer een stateless 'GameOfLifeEngine'.
   b) Koppel de engine aan de bestaande GUI code.
   c) Implementeer een stateful 'GameOfLifeEngine'.

In afbeelding 6.1 is te zien hoe de verschillende onderdelen in elkaar zitten. De 'GameOfLife' interface wordt geïmplementeerd, door de verschillende 'engines'. Als uitgangspunt heb ik als eerste een 'Procedural Engine' gemaakt, deze engine maakt geen gebruik van een Rule engine.

Uiteindelijk is er voor elk product een evaluatie geschreven.




Figuur 6.1.: Concept game of Life

### Criteria

Hieronder volgt een lijst met technische criteria:

**Installatie** Hoe complex is de installatie: Is er een eclipse plugin aanwezig? Hoe lastig de installatie is op nieuwe systemen? Werkt alles direct of is er nog meer configuratie nodig?

**Tools** Hoe goed zijn de bijgeleverde tools: Denk hierbij aan een eclipse plugin of een stand-alone tool.

**Definiëren van rules** Hoe makkelijk is het om nieuwe rules aan te maken: Is het aanmaken intuïtief? Hoe ingewikkeld is de DSL als deze er is?

**Java integratie** Hoe makkelijk is het om met Java te integreren: Is de API duidelijk? Als er code gegenereerd wordt is de interface ook duidelijk?

**Ontwikkeling** Hoe groot is het gebruiksgemak: hoe makkelijk is het om in te ontwikkelen? Kan een ontwikkelaar er snel en eenvoudig gebruik van maken? Kan een minder technische domein expert er eventueel mee over weg?

**Stabiliteit** Zijn de tools stabiel: Crasht het vaak met gevolg van verlies van data? Indien de tools gebruik maken van een eclipse plugin, blijft de rest van eclipse nog functioneren?

**Complexiteit** De complexiteit in het algemeen: Hoe complex zijn de tools, de DSL etc.?

**Duidelijkheid documentatie** Is de documentatie volledig en duidelijk: Staan de 'code snippets' vol met fouten? Is de tekst duidelijk te lezen? Is er een goed balans tussen voorbeelden en theorie?

**Support** Hoe goed is de support: Is er een actief forum? Hoe snel reageert de support van het product indien aanwezig?

**Snelheid** Hoe snel is de Rule engine: Dit doelt op de snelheid waarmee de Game of Life simulatie werkte. Hier wordt niet veel waarde aan gehangen het zijn geen officiële benchmarks en het programma is waarschijnlijk sub-optimaal geschreven.

### 6.5.2. Productevaluatie

In de bijlage C is een uitgebreid overzicht te zien van de productevaluatie van de verschillende producten. Deze zijn beoordeelt op de criteria die hierboven beschreven zijn.

### 6.5.3. Evaluatie: Tweede shortlist

Het tweede deel van het onderzoek is iets anders gelopen dan het eerste deel. Het was aanvankelijk het idee in het plan van aanpak om twee shortlists te maken. Ik heb het kiezen van het definitieve product en een technische evaluatie van de producten samengevoegd tot een enkele fase. Deze beslissing is genomen omdat nadat de producten waren geëvalueerd er redelijk snel een beslissing kon worden gemaakt. De eisen waren reeds geïnventariseerd in voorgaande fases, dit hoefde niet meer te gebeuren, na een enkele bespreking kon er al een beslissing gemaakt worden.

Waar ik tegen aan ben gelopen is het slechte functioneren van het Open Rules product. Ik heb na wat frustratie zelf de afweging gemaakt of het nog nut had om deze verder te testen. Ik besloot dat dit niet zo was. Deze beslissing heb ik overlegd met mijn bedrijfsbegeleider en vervolgens hebben we gezamelijk besloten om dit product links te laten liggen.

De complexiteit van het testen was toch groter dan ik had verwacht. Daarom moest goed opgelet worden om binnen de tijd te blijven en dezelfde hoeveelheid tijd te besteden aan elk product. Vooral het inlezen op de materie duurde langer dan verwacht, hierdoor kwam ik bij het testen soms in tijdnood.

### 6.5.4. Conclusie

Op Open rules na waren de overige producten prima te gebruiken. De producten die naar mijn mening het interessantst uit de test kwamen zijn Drools en Visual Rules. Van beide kan worden gezegd dat het degelijke producten zijn die perspectief bieden. Jess lijkt mij wat minder geschikt, het is moeilijker om te leren en biedt net wat minder dan de andere twee producten. Het is zeker geen slecht product en zal in een academische omgeving eventueel de betere keuze kunnen zijn.

## 6.6. Beantwoorden van de onderzoeksvraag

### 6.6.1. Antwoord op de onderzoeksvraag

De onderzoeksvraag luidde als volgt: **"Wat is de meest geschikte Rule engine voor Sogyo?"**. Hier wordt later op terug gekomen, eerst wil ik de deelvragen beantwoorden. De antwoorden op de deelvragen zijn op te maken uit verschillende delen van de scriptie, hieronder volgt een opsomming:

1. **Welke Rule engines zijn er?** Er bestaat een groot scala aan verschillende Rule engines. De keuze wordt wat kleiner als er naar Java integratie wordt gekeken, producten van Microsoft vallen dan af. In de scriptie is in sectie 6.3 op pagina 33, een lijst met huidige producten die op de markt zijn te vinden. Hier zijn grote producten zoals 'Blaze Advisor' express uitgelaten omdat deze voor Sogyo niet intressant zijn. Toch heb ik hier wel twee grote producten meegenomen namelijk: 'Visual Rules' en 'ILog JRules' waarvan Visual Rules later nog is onderzocht. Ik had anders het gevoel dat de diversiteit verloren zou gaan

2. **Wanneer en waarom is een Rule engine geschikt?** Duidelijk is dat een Rule engine zeker niet in alle situaties geschikt is. Het moet beschouwd worden (net zoals vele andere producten, talen etc.) als een middel om tot een oplossing te komen en niet als de oplossing zelf. Zoals er voor het schuren van hout, geen hamer gebruikt wordt. In welke situaties een Rule engine wel geschikt is, is te lezen in sectie 4.5.

3. **Wat voor consequenties heeft een Rule engine op Domain Driven Design?** Als een Rule engine op de juiste wijze gebruikt wordt, is de Rule engine een goede toevoeging voor het model. Als het model niet geschikt is voor gebruik met een Rule engine, is het gebruik ervan sowieso af te raden. Indien het model wel geschikt is, moet ervoor gezorgd worden dat de Rule engines aan dezelfde eisen wordt onderworpen als de rest van het model. Een Rule engine is in een Domain Driven Design project deel van het model.

   Tijdens het opstellen van regels moet ervoor gezorgd worden dat deze klein en overzichtelijk blijven. De globale complexiteit mag hoog zijn, maar per regel moet de complexiteit laag zijn. Hier leent een Rule engine zich goed voor, deze probeert af te dwingen dat regels niet te complex worden, anders worden regels moeilijk te onderhouden.

   Zorg ook dat een Rule engine de taak blijft doen waarvoor hij bedoeld is. Kijk dus goed naar wat een Rule engine moet doen en dat een Rule engine onderdeel blijft van het systeem. Een Rule engine mag dus niet dermate complex zijn dat het een systeem op zich is.

   Over dit onderwerp wordt later meer in de scriptie gesproken, als het antwoord op deze vraag niet duidelijk is zorgen hoodstuk 7 en 8 waarschijnlijk voor de nodige verduidelijking.

4. **Zijn er Rule engines die rekening houden met Domain Driven Design?** Nee. Dit ligt meer aan hoe een Rule engine wordt gebruikt, dan aan de Rule engine zelf. De interfacing is belangrijker dan de Rule engine. Het is daarom niet de taak van de Rule engine om rekening te houden met Domain Driven Design, maar van de interface naar de Rule engine toe. Als er gebruik wil worden gemaakt van een Rule engine moet de Rule engine in het model passen. Dit vereist een bepaalde manier van de Rule engine gebruiken, hiermee moet rekening gehouden worden door de ontwikkelaars, niet door de Rule engine.

5. **Welke eisen stelt Sogyo aan een Rule engine?** Door interviews en informele gesprekken met medewerkers ben ik erachter gekomen welke eisen belangrijk zijn. Dit zijn de volgende eisen:
   - Het product dient makkelijk te zijn voor de developer, niet de business expert.
   - Het product moet een niet al te dure licentie te hebben.
   - Programmeurs die met Java en C# bekend zijn moeten er, zonder al te veel moeite, mee kunnen werken.
   - Een Rule engine project moet makkelijk overdraagbaar zijn.
   - Gelieve voldoende documentatie beschikbaar.
   - Het product moet kunnen interfacen met zowel Java of .Net.

6. **Zijn er in voorgaande projecten elementen geweest die vervangen konden worden door een Rule engine? Zo ja, welke?** Naar deze vraag wordt in de implementatiefase nog in detail gekeken. Projecten die in de aanmerking kwamen gedurende de interviews waren project Norma en in mindere mate project de Leeuw.

#### Uiteindelijk gekozen product

Nadat deze vragen beantwoord zijn en de verschillende producten getest waren, kon er een definitief product worden gekozen. Waardoor de onderzoeksvraag is beantwoord.

Het uiteindelijke gekozen product is de **Drools** Rule engine. Drools biedt een mooie ontwikkelmethodiek en omgeving. Drools scoorde goed in beide tests. Het is een product wat degelijk is en ook in de praktijk gebruikt is, door grote bedrijven zoals bijvoorbeeld FedEx. Drools is makkelijk om in te ontwikkelen en de licentie is ook aantrekkelijk. Alle belangrijke aspecten zijn door Drools gewaarborgd zoals o.a. goede documentatie, activiteit en gemak van gebruik. De actieve mailing list is ook een plus punt, zo kunnen er eventuele vragen redelijk vlot beantwoord worden. Al met al is Drools dus een goed product wat zich volgens de ontwikkelaars in productie heeft bewezen, zodoende kon er met vertrouwen voor het Drools product worden gekozen.

Visual Rules kwam ook in de aanmerking. Helaas zorgde de dure licentie ervoor dat het product uiteindelijk geen optie was. Daarbij kan er functionaliteit gebruikt wordt van andere producten uit het Drools platform, waardoor een deel van de unieke functionaliteit gevonden in Visual Rules, vervangen kan worden door het Drools platform. De Tools in Visual Rules zijn beter, toch haalt dit het niet bij de voordelen van Drools.

Ik ben tot deze conclusie gekomen samen met de bedrijfsbegeleider, we hebben in een gesprek de opties besproken en zijn toen op Drools uitgekomen.

## 6.7. Conclusie

Ik ben van mening dat mijn onderzoeksvraag goed beantwoord is. Ik heb op alle deelvragen kunnen antwoorden en uiteindelijk is er een geschikt product voor Sogyo gekozen. Aan de keuze van Drools als product hangen wel enige nadelen die belangrijk zijn te begrijpen voordat er voor Drools gekozen wordt in een project.

Hieronder volgt een opsomming van de nadelen:

- Het Drools product heeft kennis nodig om gebruikt te worden, meer dan Visual Rules.
- Omdat het een open-source product is, moet voor commerciële support JBoss Rules gebruikt worden.
- Drools heeft een redelijke leercurve, lager dan die van Jess maar hoger dan die van Visual Rules.
- Het opstellen van regels moet zorgvuldig worden gedaan, een regel die resulteert in 2 miljoen activaties is inefficiënt.
- Achterliggende kennis is noodzakelijker voor het gebruik van Drools, dan bij Visual Rules.

Een aantal van deze punten zijn voor meerdere Rule engines van toepassing, desalniettemin veranderd dit niet het feit dat dit bij Drools net zo zeer geldt.

# 7. Proof of Concept implementatie

## 7.1. Inleiding Proof of Concept

In dit hoofdstuk wordt beschreven wat het Proof of Concept is en op welke wijze het project is uitgevoerd.

Het Proof of Concept is een implementatie van een op Rule engine gebaseerd product. Daarmee is het Proof of Concept het uitvoerende deel van het onderzoek, het uitgekozen product wordt in deze fase in de praktijk gebruikt.

### 7.1.1. Doel

Het doel van het Proof of Concept, is de toepasbaarheid bewijzen van een Rule engine voor Sogyo. Het product moet aan de volgende eisen voldoen:

- Het Proof of Concept moet binnen de Sogyo ontwikkelmethodiek passen.
- Het Proof of Concept moet vergelijkbaar zijn met projecten die Sogyo normaal uitvoert.
- De verschillen tussen de twee implementaties moeten aantoonbaar zijn.

Om aan deze eisen te voldoen waren de volgende beslissingen genomen:

- Het Proof of Concept wordt op een DDD (Domain Driven Design) wijze opgezet.
- Het Proof of Concept zal gebruik maken van een bestaand project als casus.
- Het project dat uitgekozen wordt, moet zich lenen voor een vergelijking van de verschillen.

### 7.1.2. Norma

Een project was nodig als casus, dit project is uitgekozen in overleg met de bedrijfsbegeleider en de projectmanager binnen Sogyo. Van te voren zijn er een aantal projecten bekeken, waar voornamelijk de afweging was of het baat zou hebben om voor dat project een Rule engine te gebruiken. Er is echter wel in het achterhoofd gehouden, dat het Proof of Concept de toepasbaarheid bewijzen voor Sogyo als doel heeft, daarom was een reeds uitgevoerd project de beste casus.

Het project wat uiteindelijk voor het Proof of Concept is uitgekozen is project Norma. Van de door ons bekeken projecten lijkt Norma het meest geschikt, een project moet namelijk redelijk ingewikkelde bedrijfslogica bevatten om het gebruik van een Rule engine te kunnen verantwoorden. Norma leek het meest aan deze eis te voldoen.

Het Norma project is in C# geschreven, het lijkt wellicht een vreemde keuze om een project te kiezen dat niet in Java is geschreven. Een aantal redenen zijn aanwezig waarom deze keuze is gemaakt: Het was belangrijker een project te kiezen, dat werkelijk door Sogyo is uitgevoerd dan een project die perfect aansluit op een Rule engine. Toch moet er wel enige aansluiting zijn en daarvoor leek Norma, toevallig een C# project, het meest geschikt. Sogyo gebruikt zowel Java als C# in grote mate, zowel in de masterclass[1] als in de praktijk.

### Over Norma

Hieronder volgt een korte inleiding over Stichting Norma en het huidige product:

> "Stichting Naburige Rechtenorganisatie voor Musici en Acteurs, kortweg NORMA, is een incasso- en verdeelorganisatie voor uitvoerend kunstenaars. Maar NORMA is meer dan dat. NORMA behartigt de belangen van musici, acteurs en andere performers op het gebied van naburige rechten. Daarnaast is NORMA zowel nationaal als internationaal actief om wetgeving op het gebied van naburige rechten te verbeteren.
>
> NORMA vertegenwoordigt alle uitvoerend kunstenaars bij de collectieve exploitatie van hun naburige rechten. Deze collectieve exploitatie vindt plaats om puur praktische redenen. Een uitvoerend kunstenaar kan zelf onmogelijk bijhouden of zijn werk is thuisgekopieerd of is uitgeleend door een bibliotheek, terwijl hij daarvoor wel recht heeft op een financiële vergoeding. NORMA kan dit wel en is zelfs voor dit doel opgericht. NORMA int de vergoedingen ten behoeve van de bij haar aangesloten uitvoerende kunstenaars en keert deze aan hen uit."[10]

Het Norma systeem is een systeem om het uitbetalen vanuit verschillende 'Budgetten' te automatiseren. Thuiskoppieheffing is bijvoorbeeld een *Budget*, Norma rekent dan uit wie en hoeveel de artiesten uitbetaald krijgen. Hierbij zijn verschillende factoren die meewegen, zoals bijvoorbeeld in/niet inbeeld factoren en de *Score's* van een *Werk*, dit bepaalt het percentage van een budget wat een *Werk* (en vervolgens een *Performer*) ontvangt.

Norma is van de projecten die bekeken zijn de meest geschikte. Het bevat redelijk complexe logica met een aantal uitzonderingen. Door een projectlid die momenteel aanwezig bij Sogyo is, gaf aan, dat de hoeveelheid uitzonderingen iets was waar in dat project tegen aangelopen werd.

### Complexiteit

Hieronder volgen een aantal factoren die de complexiteit bepaalden van het Norma project:

[1] De verplichte opleiding die alle werknemers moeten volgen.

- **Grote hoeveelheden data**. Het project werkte met een grote hoeveelheid data die in één keer verwerkt werden. Tussenstanden worden niet bijgehouden, daardoor gebruikte de applicatie vaak te veel geheugen. Door dit feit is veel van de logica verplaatst naar de Database in de vorm van stored procedures.
- **Veel uitzonderingen**. er zitten veel uitzonderingen in de logica, waardoor er met veel dingen rekening moet worden gehouden.
- **Complex domein**. Het domein is ingewikkeld en er zijn veel interne koppelingen.

Het voornaamste doel van het Proof of Concept is om de logica op een wat toegankelijkere wijze op te zetten, zodat de logica overzichtelijk en duidelijk is. Een Rule engine zorgt niet automatisch voor efficiënter gebruik van de data, een grote snelheidswinst is niet een van de voornaamste doelen van een Rule engine.

Wat een Rule engine wel kan, is de uitzonderingen beter in kaart brengen, op een duidelijkere en declaratieve wijze. Een Rule engine kan ook makkelijker tussenstanden bijhouden en verandering d.m.v. regels beter doorvoeren.

### 7.1.3. Verwachting

Hieronder volgt een verwachting die voor het ontwikkelen van het Proof of Concept is opgesteld:

> Het is van te voren niet goed in te schatten wat de uitkomsten van het Proof of Concept zullen zijn. Er zijn verschillende scenario's mogelijk. Van te voren is duidelijk dat een Rule engine alleen in specifieke situaties geschikt is, dit heeft de onderzoeksfase aangetoond, het is nog niet duidelijk of Norma in deze specifieke situatie past. De vraag of een Rule engine in een DDD aanpak past, blijft aanwezig.
>
> Ik verwacht dat een Rule engine wel in een DDD aanpak kan passen.Eventueel moet er op een andere manier naar het domein gekeken worden, maar dit moet wel te realiseren zijn. De vraag of een Rule engine in zijn totaliteit geschikt is voor Sogyo blijft nog onduidelijk. Van te voren wordt wel verwacht dat er aanpassingen nodig zullen zijn in het Norma proces om op een natuurlijke manier samen te kunnen werken met een Rule engine.

### 7.1.4. Kritieke punten

Tijdens het bouwen van het Proof of Concept zijn drie punten aanwezig die voortdurend in het achterhoofd gehouden moeten worden:

1. Het Norma domein begrijpen.
2. Het project op een Domain Driven Design manier opzetten.
3. Het integreren van een Rule engine.

Tijdens de implementatie moest constant aan deze drie punten gewerkt worden. Deze punten waarborgen het behalen van de eisen die opgesteld zijn in sectie 7.1.1.

### 7.1.5. Projectaanpak

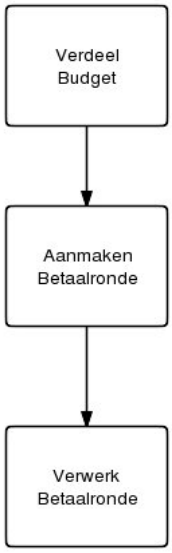
#### Inleiding

Omdat het originele Norma project is geschreven in C#, was het noodzakelijk dat delen van het project 'geport' moesten worden naar Java code, aangezien dat de taal is waarmee Drools werkt. Er is voor gekozen om een gedeelte van het domein te destilleren, zodat alleen dat wat noodzakelijk is om het proces globaal na te bootsen gebruikt zal worden

Getracht is om deel voor deel het Norma proces na te bouwen. Per deel het bijbehorende domein op te bouwen en zo stapsgewijs de functionaliteit te implementeren.

#### Proces

Het proces van Norma bestaat uit drie verschillende grote processen. In figuur 7.1 is een overzicht te zien.




Figuur 7.1.: Globaal overzicht van het proces van Norma

Deze drie processen gebruiken zowel overlappende domeinobjecten als domeinobjecten specifiek voor dat proces. Het project is dan ook aan de hand van deze fase's opgedeelt.

### 7.1.6. Projectmethodiek

#### Sprints

Ik had besloten om met behulp van sprints het project op te delen. Om zo een soortgelijke projectmethodiek te hanteren als binnen Sogyo. Omdat het project door een enkel persoon wordt uitgevoerd, is ervoor gekozen niet een hele projectmethode te gebruiken zoals

bijvoorbeeld Scrum. Sprints zijn opgedeeld in periode's van één tot twee weken. Een sprint bevat de volgende fase's:

1. Analyse/Ontwerp. Wat moet er gebeuren? Wat wordt er gemaakt?
2. Implementatie. Het implementeren van de taken.
3. Evaluatie. Wat moet er in de volgende sprint anders? Schuift er iets door?

Het idee achter de sprints is dat er na elke sprint een demobaar product is. Hierdoor moet de GUI in elke sprint worden bijgewerkt. De sprints worden opgedeelt over de verschillende onderdelen van het Norma proces (zie 7.1). De eerste stap wat in elk onderdeel wordt gedaan is het domein bijwerken, dit domein wordt vervolgens gekoppeld met de Rule engine en de GUI.

Aan het begin van de implementatiefase is uitgegaan van de volgende sprints:

1. **Verdeel Budget**
   a) Basis applicatie opzetten met CRUD functionaliteit.
   b) Rule engine integereren in het project.
2. **Persistentie toevoegen.**
3. **Aanmaken Betaalronde**
   a) Domein uitbreiden inclusief GUI.
   b) Rule engine integratie.
4. **Verwerk Betaalronde**
   a) Domein uitbreiden.
   b) Rule engine integratie.
5. **Complexiteit en uitzonderingen toevoegen**

Al snel bleek dat het opzetten van een DDD project, zonder echte voorkennis, lastiger is dan gedacht. Hier kom ik later nog bij de evaluatie op terug.

#### Domain Driven Design

Er werd besloten om een domain driven design aanpak te hanteren omdat er vanuit Sogyo veel met deze methodiek wordt gewerkt. Het is daarom interessant om te kijken of een Rule engine in deze methodiek zal passen. Over het antwoord op deze vraag is eerder in dit document al gespeculeerd, echter uitsluitend door het in de praktijk toe te passen kan hier over een uitsluitsel over worden gegeven.

De focus ligt zoveel mogelijk op het verantwoordelijkheid geven aan het domein. Daardoor zal de Rule engine puur voor de regels gebruikt moeten worden en minder voor de logica. De reden hiertoe is simpel: de Rule engine moet gebruikt worden zoals deze in een 'echt' project ook gebruikt zal worden.

#### Tools

Voor het Proof of Concept zijn de volgende tools gebruikt:

- Drools. Het uitgekozen product uit de vorige fase van de opdracht.
- Eclipse. Als ontwikkelomgeving.
- SWT. Voor de GUI.

## 7.2. Uitwerking Proof of Concept

In dit volgende deel van de scriptie wordt toegelicht hoe het Proof of Concept in elkaar is gezet tot op het moment dat de scriptie is geschreven. In het volgende hoofdstuk van de scriptie worden een aantal conclusies getrokken, gebaseerd op de onderdelen die in dit deel van de scriptie worden besproken.

### 7.2.1. Projectopbouw

Zoals in sectie 7.1 is besproken, is dat inzicht verschaffen over de toepasbaarheid van een Rule engine het doel is van het Proof of Concept. Om dit doel te bereiken moet het Proof of Concept elementen bezitten van Sogyo projecten.

Het project wat uitgevoerd is, is een destillatie van het oorspronkelijke Norma project. Dit 'nieuwe' project is geschreven in Java met behulp van de Drools Rule engine.

#### Projectopzet

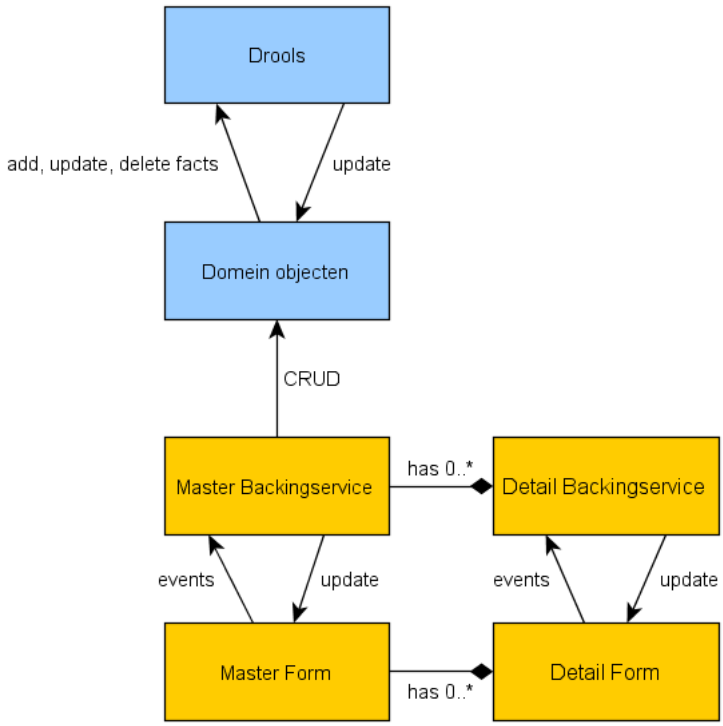
Het Proof of Concept project bestaat uit de volgende onderdelen:

**Domein model** Het belangrijkste deel van het Proof of Concept is het domein. Deze bevat de logica van het Norma domein. De domein objecten zijn gebaseerd op het oorspronkelijke C# domein. Deze beslissing is bewust genomen omdat ik het Proof of Concept dicht bij het oorspronkelijke project wilde houden. Zodoende kan er beter geëvalueerd worden of een Rule engine zonder meer in een bestaand project past. Wel is er geprobeerd een balans te vinden tussen het oorspronkelijke domein aanhouden en een aantal Domain Driven Design keuzes. Een eis was dat het project op een Domain Driven Design wijze wordt opgezet, aangezien dit bij het oorspronkelijke project niet geheel het geval was, heb ik een aantal Domain Driven Design elementen toegevoegd.

**Drools** Drools is de Rule engine die gebruikt is voor het Proof of Concept project. Drools zal deel uitmaken van het domein model. Dat houdt in dat Drools en het de domein objecten elkaar direct kunnen aanroepen en aanpassen. In Domain Driven Design wordt de nadruk gelegd op het lagen model en de regels die daarbij horen. Één daarvan is dat een lagere laag de hogere nooit mag aanroepen. Concreet zou dit betekenen dat als de Rule engine zich niet in het domein model bevindt, de domeinobjecten geen kennis mogen hebben van de Rule engine. Het is logisch dat

de Rule engine deel van het domein is, aangezien deze de domein regels bevat. Normaal worden deze regels in objecten gemodelleerd. In dit geval is dat niet zo; regels worden in een extern systeem gemodelleerd. De Rule engine is vooral gebruikt voor de business regels. Tijdens het toevoegen van een regel aan de Rule engine is steeds de vraag gesteld of het een deel van het proces is of dat het daadwerkelijk een regel is.

**GUI** Als GUI toolkit is er SWT gebruikt. Deze keuze is gemaakt omdat ik voornamelijk met Swing heb gewerkt en wilde proberen hoe SWT in elkaar zit. De GUI is voor een agile project belangrijk, aangezien er bij elke sprint iets demobaars opgeleverd moet worden. De GUI is in het Proof of Concept eenvoudig, het bestaat uit verschillende 'master-detail' schermen die elke een 'master' object hebben en eventueel meerdere detail objecten. Er zit geen logica in de schermen zelf, elk scherm heeft een 'backingservice' wat fungeert als een controller object.




Figuur 7.2.: Overzicht Proof of Concept onderdelen

In figuur 7.2 is te zien hoe de componenten in het project communiceren. Kleuren groeperen de lagen, de blauwe blokken zijn het domein en de oranje de User interface. De domein objecten en de Rule engine communiceren direct met elkaar, door het toevoegen van entiteiten en updates worden er regels geactiveerd, als er regels afgevuurd worden past de Rule engine de objecten direct aan.

De GUI bestaat uit meerdere master-detail forms en meerdere 'master-backingservices' met 'detail-backingservices'. De forms zijn in de praktijk één op één gekoppeld met de backingservices, maar dit hoeft in theorie niet. Iedere 'backingservice' met bijbehorende 'form' heeft de verantwoordelijkheid voor het bijhouden van een enkel domein object. De 'form' heeft de verantwoordelijkheid voor het weergeven van de data. De 'backingservice' heeft de verantwoordelijkheid voor het aanpassen van de data en door te sturen naar de 'form'.

Dit is in vogelvlucht een overzicht van de verschillende onderdelen van het Proof of Concept project. In de volgende delen van dit hoofdstuk, zal meer inzicht verschaft worden over hoe Domain Driven Design en Rule engines in deze onderdelen terug komen.

### 7.2.2. Domain Driven Design

Domain Driven Design speelt een prominente rol in het Proof of Concept project. Vanaf het begin van het project maakt het domein model gebruik van Domain Driven Design technieken. Hieronder worden een aantal Domain Driven Design aspecten toegelicht van het Proof of Concept project:

#### Opbouw domein model

Het domein model is in meerdere iteraties opgebouwd. Welke delen van het domein nodig waren is aan de hand van de fases (zie figuur 7.1) besloten. Het voornaamste doel was het proces nabootsten, zonder extra's. Een aantal domein objecten uit het originele Norma project waren uiteindelijk niet nodig. Voor het opbouwen van het domein model zijn de volgende bronnen gebruikt:

- **Raymond Tukkers** Raymond is een programmeur die aan het originele Norma project heeft gewerkt, dit was eigenlijk de domein expert in het Proof of Concept project. Aangezien hij meer van het domein wist dan iemand anders in Sogyo, is veel met hem in samenspraak gedaan. Samen met Raymond is gekeken welke objecten nodig zijn uit het originele model en wat de functie was van veel van die objecten. Omdat een model al aanwezig was, waren er een aantal termen al gecristalliseerd, zodoende bestond er al een soort 'ubiquitious language'. Tijdens de communicatie is actief in deze 'ubiquitious language' gesproken.

- **Norma Vuik II specificatie** In het originele project, is een groot gedeelte van het proces geëncapsuleerd in stored procedures vanwege performance redenen, deze zijn lastig om te begrijpen. Daarom is voornamelijk gebruik gemaakt van de specificatie, omdat dit een 'schonere' specificatie is. De specificatie is door de klant opgesteld en is gebruikt voor de ontwikkeling van de originele Norma applicatie. Deze specificatie

is technisch en uitermate gedetailleerd. Zo wordt zelfs opgedragen welke tekst er *precies* in verscheidene pop-up vensters moet komen te staan. Hierdoor is het zeer compleet en kan er veel informatie uitgehaald worden. Anderzijds wordt het model hierdoor star: er kan niet langzaam ondekt worden wat het beste werkt in het model, immers is dit in de exacte werkwijze al gespecificeerd.

### Domein model

Het domein model is zo goed mogelijk op een domein driven design wijze opgezet. Als uitgangspunt is het domein model van het orginele project gebruikt. Het originele domein is niet één op één overgenomen. Er is voornamelijk gebruik gemaakt van de orginele klasse hiërarchie. In het originele domein model is alles gekoppeld, van deze koppelingen is een groot deel achterwege gelaten. Om het model specifieker te maken zijn er extra objecten (Value Objects) geïntroduceerd.

Hieronder worden een aantal technieken en 'patterns' besproken die in de domeinobjecten zijn gebruikt, waarom deze patterns juist belangrijk zijn voor een goed Domain Driven Design valt buiten de scope van deze scriptie.

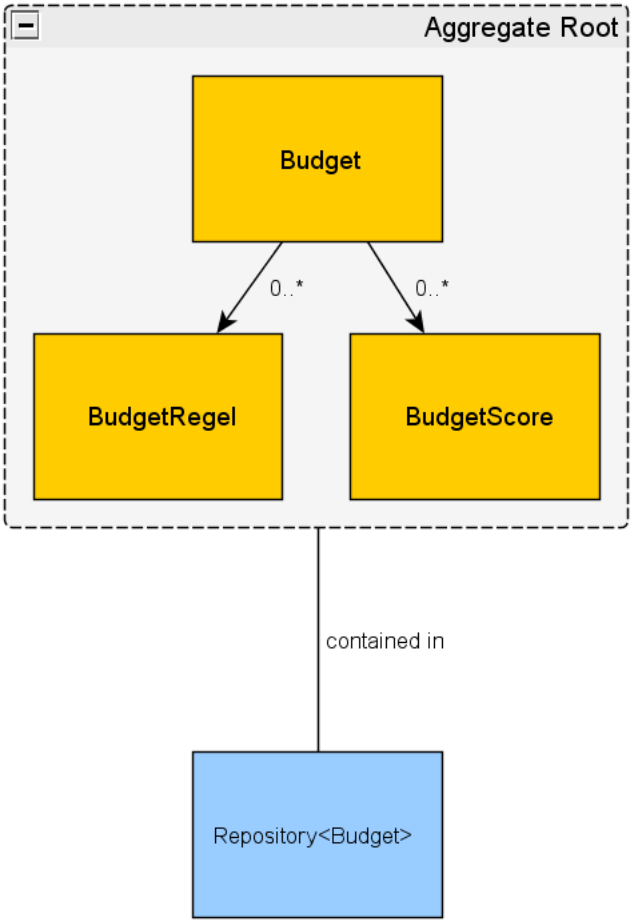
**Entity en Value Objects** Een 'entity' in een Domain Driven Design model zijn objecten waar de identiteit van moet worden bijgehouden. Dit zijn objecten als: Persoon, Order, Product etc. Anderzijds zijn Value objects uitwisselbaar. Als de eigenschappen van twee objecten hetzelfde zijn, zijn twee objecten identiek. De identiteit is onbelangrijk. Denk hierbij aan objecten als: Adres of Geld. Immers, als iemand betaalt in de winkel voor een product met een tien euro biljet wordt niet gevraagd: 'Met welk tien euro biljet was dat?'. Daarentegen, als iemand je verteld dat hij laatst Jan heeft gezien, en je kent er meerdere, dan ben je wel geintreseerd welke 'Jan' dat is. In het eerste geval ben je niet geïntresseerd in de identiteit, in het tweede geval wel.

In het Proof of Concept hebben alle entiteiten als superklasse de abstracte klasse: *Entity*. Alle value objects hebben als superklasse de abstracte klasse: *ValueObject*. De *Entity* klasse geeft alle entiteiten 'PropertyChangeSupport' dit wordt o.a. gebruikt voor communicatie naar Drools. De *ValueObject* klasse heeft geen body maar zorgt ervoor dat de desbetreffende objecten voldoen aan de specificatie van een Value Object.

De distinctie, tussen entiteiten en value objects, wordt gemaakt in Domain Driven Design omdat de semantische betekenis van de objecten anders zijn. In Domain Driven Design wordt aangeraden zo specifiek mogelijk te zijn.

**Aggregate Roots** Aggregate roots zorgen voor een extra semantische laag in het objectenmodel. Door een aantal entiteiten met een Aggregate root te encapsuleren, onstaan er specifiekere relaties tussen de enititeiten. In Domain Driven Design wordt alleen toegestaan de entiteiten via een Aggregate root te benaderen.

In figuur 7.3 zijn een aantal objecten te zien, dit zijn een aantal entiteiten uit het project. Deze objecten behoren allen tot dezelfde Aggregate Root, namelijk: *Budget*. *BudgetRegel* en *BudgetScore* zijn alleen toegankelijk vanuit *Budget*.




Figuur 7.3.: Aggregate Root met Repository

Als er een BudgetRegel en BudgetScore aan Budget toegevoegd worden, resulteert dat in de volgende code:

Listing 7.1: Aggregate Root encapsulatie

```
//Haal Budget op
Budget budget = BudgetRepository.get(1);

//Maak nieuwe BudgetRegel aan
BudgetRegel regel = budget.newBudgetRegel("Regel 1", 500.0);
//Koppel budget aan werk
BudgetScore score = budget.koppelWerkAanBudget(werk);

//Sla Budget op
BudgetRepository.save(budget);
```

In de code is te zien dat er een methode in de Aggregate root *Budget* zit voor het aanmaken van een nieuwe *BudgetRegel*. Een *BudgetScore* is een koppeling tussen *Budget* en een *Werk* daarom heet de methode ook 'koppelWerkAanBudget'.

**Repositories** Een Repository is een manier om te abstraheren hoe data wordt opgeslagen. Een Repository ziet er van de buitenkant uit als een collectie, maar de dataopslag kan op verschillende manieren gebeuren. ORM is een opslagtechniek, JDBC is een ander, vaak wordt er tijdens testen een in-memory opslagmechanisme achter gezet.

In de code in listing 7.1, is te zien hoe een Repository gebruikt wordt.

### 7.2.3. Rule engine

In het Proof of Concept speelt de Rule engine natuurlijk een erg belangrijke rol. De reden waarom het Proof of Concept is opgezet is om de bruikbaarheid van een Rule engine te testen. Er werd voor gewaakt om de Rule engine niet te misbruiken, tijdens het Proof of Concept is goed gekeken of iets daadwerkelijk in de Rule engine hoort. In deze sectie van het hoofdstuk zullen ook wat voorbeelden gegeven worden, waar is besloten dat iets niet in de Rule thuis hoort.

#### Ontdekken van Regels

In het originele Norma project zitten de regels vooral in de stored procedure code. Dit is uiteraard niet ideaal, de regels komen niet duidelijk naar voren en er moeten rare constructies worden bedacht om een aantal uitzonderingen in te kunnen bouwen. Het was dan ook niet gemakkelijk om deze stored procedure code te gebruiken om de regels te destilleren. Hiervoor is van dezelfde bronnen gebruik gemaakt voor het opbouwen van het objectenmodel.

Tijdens het lezen van de Norma specificatie is vooral geprobeerd het proces te doorzien: Welke stappen moeten genomen worden en welke regels horen daarbij? Er is vooral gezocht naar de woorden: "Als" en "Dan". Uit de taal is vaak al te herkennen of het om een imperatieve procedure gaat: "Lees de data, Sla het op in de database", of een declaratieve regel: "Als de data aan deze voorwaarden voldoet dan is deze juist".

Geprobeerd is om net zoals tijdens het bouwen van het objectenmodel, alleen de regels te destilleren die nodig zijn om het basisproces na te bootsen.

Tijdens de communicatie met Raymond Tukkers, moest ook vaak een stap terug worden gedaan en worden bedacht welke Regels in het gesprek naar voren kwamen. Het hielp erg om met Raymond te kunnen praten, over het Norma proces te kunnen redeneren en zo het domein beter te leren begrijpen. Het lastige was om uit het oude model te stappen en vervolgens op een andere wijze over het model na te denken, een model waar Regels en proces technisch gescheiden zijn.

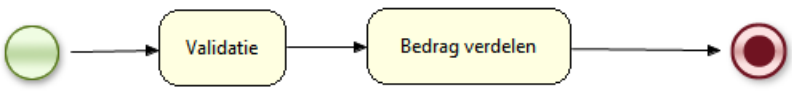
### Ontwikkeling van Regels

Vroeg in het implementatieproces is besloten de Rule engine een onderdeel van het domeinmodel te maken. De regels in een Rule engine horen in het domein thuis, daarom mag de Rule engine de domeinobjecten direct aanpassen en wordt vanuit het domein de Rule engine aangesproken.

Het proces, zoals afgebeeld in figuur 7.1, speelt een centrale rol voor Norma. De Norma applicatie bestaat uit een aantal administratieve schermen. Deze schermen zetten alles klaar voor een of meerdere gedeelte van het proces, bijvoorbeeld het verdelen van het geld over het budget. Daarom is besloten om ook een ander deel van Drools te gebruiken: Drools Flow.

Drools is zoals eerder vermeld een platform met meerdere componenten, de Rule engine is daar één van. Drools is begonnen als een Rule engine en de andere componenten zijn er later bijgekomen. In versie 5.0 zijn al deze andere componenten, net zoals de Rule engine, 'first-class citizens' geworden. Wat dit in houdt is dat Drools zich tegenwoordig meer als een Knowledge engine profileert. De Rule engine speelt vooralsnog wel de grootste rol en komt ook bijna in alle andere producten terug.

**Drools Flow** Norma forceerde dat regels in een bepaalde volgorde afgevuurd worden. Dit is te regelen in de Rule engine met 'agenda groups', echter kan er ook gebruik gemaakt worden van een RuleFlow. Een RuleFlow kan een bepaalde flow van een proces definiëren. In 7.4 is een deel van een proces in Norma te zien, de tweede nodes zijn groepen waar regels onder vallen. Van deze flow modellen is gebruik gemaakt tijdens de ontwikkeling van het Proof of Concept, zo werd duidelijker in welke volgorde de regels uitgevoerd moesten worden.




Figuur 7.4.: Een Drools Flow model

### 7.2.4. Regels in het Proof of Concept

In deze sectie worden een paar voorbeelden gegeven van regels die gebruikt worden in het Proof of Concept. Het voornaamste doel in dit onderdeel is om een beeld te geven waarom bepaalde regels zijn gekozen en de motivatie achter de keuzes.

#### Regels niet in de Rule engine

Een van de doelen was om de Rule engine te gebruiken voor voornamelijk de business regels, de regels moeten thuis horen in de Rule engine. Daarom is er besloten om bepaalde regels niet in de Rule engine te zetten.

Een voorbeeld hiervan is de regel voor het bepalen van een huidige categorie. Aan een Werk kunnen drie categorieën hangen: aangeleverde, aangepaste en Norma categorie. De huidige categorie wordt bepaald aan de hand van welke categorieën aanwezig zijn:

1. Als alleen de **aangeleverde categorie** aanwezig is, dan is de aangeleverde categorie de huidige categorie.
2. Als de **aangepaste categorie** aanwezig is, dan is de aangepaste categorie de huidige categorie.
3. Als de **norma categorie** aanwezig is, dan is de norma categorie de huidige categorie.

Dit is in essentie een regel, de huidige categorie wordt namelijk bepaald door de aanwezigheid van een aantal velden. De vraag is: Hoort deze regel thuis in de Rule engine? Ik vind van niet, om de volgende redenen:

- Er zijn drie aparte regels nodig voor de drie cases.
- De regel zal niet vaak of nooit veranderen.
- Huidige categorie is zelf een calculated field, dus de oplossing hoort in de 'getHuidigeCategorie' methode. Om dit mogelijk te maken moet de regel afgevuurd zijn voordat iemand deze methode aan roept.
- Het is een enkele loststaande Regel, geen onderdeel van het beslissingsproces.
- De algoritmische oplossing is simpel.

Wat ik de belangrijkste afweging vind is de semantische vraag: Is deze regel deel van het beslissingsproces? In mijn opinie niet, eigenlijk is het niks meer dan een property. Het is daarom, in mijn opinie, correct om deze Regel in een property te encapsuleren.

#### Regels in de Rule engine

In deze sectie worden een aantal voorbeelden gegeven van regels die wel gebruikt worden in het Proof of Concept.

**Validatie** Een soort regels die in de Rule engine zitten zijn validatie regels, deze regels zijn deel van het proces. Allereerst wordt de data gevalideerd voordat de rest van het proces kan worden doorlopen. Een voorbeeld hiervan is bij de betalingen: voordat er uitbetaald kan worden, moeten eerst alle gegevens juist bevonden worden. Validatie is daardoor een onderdeel van het Norma Proces.

Als volgt is een van de validatie regels afgebeeld, welke bepaalt dat een budget valide is, mits deze gefiatteerd is:

Listing 7.2: Een validatie regel

```
rule "Budget niet gefiateerd"
  ruleflow-group "valideer"
  when
    betaalronde : Betaalronde( status == BetaalrondeStatus.TE_BETALEN )
    budgetBetaalronde : BudgetBetaalronde( betaalronde == betaalronde,
                        budget.gefiatteerd == false )
  then
    betaalronde.isValid(false);
end
```

**Procesregels** Andere regels zijn gebruikt om het proces te ondersteunen. De volgende regel bepaalt dat voor elk *Werk* wat in een *Betaalronde* voorkomt een totaalscore berekening mag worden gemaakt. Dit gebeurd door een *TotaalScoreWerk* object aan het systeem toe te voegen. *TotaalScoreWerk*, een Value Object, heeft een aantal publieke properties die vervolgens weer uitgelezen kunnen worden:

Listing 7.3: Een ondersteunende regel

```
rule "set totaalscore"
  ruleflow-group "verdeel-bedrag"
  when
    betaalronde : Betaalronde( status == BetaalrondeStatus.TE_BETALEN )
    budgetBetaalRonde : BudgetBetaalronde( betaalronde == betaalronde,
                                           budget : budget )
    BudgetScore( budget == budget, ref : score.werk )
    werk : Werk( this == ref )

  then
    System.out.println("Insert TotaalScoreWerk object");
    insert( RuleObjectFactory.newTotaalScoreWerk(werk, budget) );
end
```

Dit *TotaalScoreWerk* object wordt later gebruikt door een andere regel. Deze regel bepaalt of het percentage wat een hoofdrol krijgt, van het uit te betalen bedrag, kleiner dan 50% is:

Listing 7.4: Bepaald of aan het verplichte percentage wordt voldaan

```
rule "hoofdrolInBeeld < 50%"
  ruleflow-group "verdeel-bedrag"
  when
```

```
    totaalScore : TotaalScoreWerk( werk : werk )
    percentage : Double( doubleValue < 50.0)
                 from totaalScore.percentageVoorHoofdRolInBeeld()
  then
    totaalScore.voldoetAanVerplichtPercentageInBeeld(false);
    //start alternatieve betaling
end
```

De twee regels hierboven zijn ondersteunend voor het proces. Aan de hand van de regel in listing 7.4 wordt bepaald hoe de betaling verloopt, de regels maken beslissingen in het proces. In het geval dat er minder dan 50% aan de hoofdrollen wordt betaald, moet er alsnog voor worden gezorgd dat de hoofdrollen 50% van het bedrag eerlijk verdeeld krijgen. Dit wordt gewaarborgd door de regel uit 7.4.

De meeste van de regels die ik uit het domein heb gedestilleerd zijn procesregels.

**Calculatie regels** Een derde soort regel die in de Rule engine voorkomt is een calculatie regel. De onderstaande regel is daar een voorbeeld van, deze regel berekent voor elk *Budget* hoeveel geld over 'in beeld' en 'niet in beeld' moet worden verdeeld:

Listing 7.5: Verdeel bedrag

```
rule "Budgetscore verdeel"
  ruleflow-group "verdeel-budget"
  when
    b : Budget()
    score : BudgetScore(budget == b)
  then
      setBedragInBeeld(enkeleGeldProcent(b) *
                       score.getScore().getScoreInBeeld()),
      setBedragNietInBeeld(enkeleGeldProcent(b) *
                       score.getScore().getScoreNietInBeeld())
end
```

Van deze derde soort Regel is niet helemaal zeker of deze in de Rule engine thuis hoort. De regel is deel van het proces, maar is een calculatie zelf een regel? Dat er een berekening verplicht moet worden gedaan, is eventueel een regel. De calculatie zelf had eventueel beter gedelegeerd kunnen worden naar het objectenmodel. Achteraf gezien zouden deze regels misschien niet in de Rule engine gezet moeten worden.

### Samenvatting

In het Proof of Concept komen een aantal regels voor, voordat een regel in de Rule engine geplaatst werd, is eerst nagegaan of deze er wel thuis hoort.

De volgende soorten regels komen op dit moment in de Rule engine voor:

- Validatieregels.
- Procesregels.
- Calculatieregels.

Al deze regels worden gebruikt binnen het Norma proces dat in figuur 7.1 is afgebeeld. Om dit proces als 'kennis' in de Rule engine te 'vangen' is een Drools flow model gebruikt.

### 7.2.5. Evaluatie: Projectverloop

De implementatie van het Proof of Concept liep wat minder gemakkelijk dan het onderzoek. De grootste oorzaak hiervan was de verschillende complexiteiten die in de implementatiefase samen kwamen. Dit zijn de complexiteitein die al eerder in sectie 7.1.4 besproken zijn. Dit had als gevolg dat mijn sprints uitliepen en ik niet elke week iets 'demobaars' had.

#### Domain Driven Design

Het Domain Driven Design gedeelte duurde een stuk langer dan gepland, het maken van een redelijke opzet kostte bijna een maand werk. Dit zorgde ervoor dat de integratie met de Rule engine steeds vooruit geschoven werd. Doordat ik een goed Domain Driven Design project wilde opzetten, verschoof mijn aandacht naar andere zaken die niet direct een correlatie hadden met de Rule engine. Andere vragen hielde mij bezig, zoals: Moet er van DTO's (Data Transfer Objects) gebruik gemaakt worden? Wat is de beste manier om de communicatie met GUI mogelijk te maken? Welke verantwoordelijkheid ligt waar?

Met als gevolg dat ik vele malen mijn code heb moeten herschrijven, om voor mijn gevoel een degelijk model op te leveren. Volgens Evans is dit normaal, hij beschrijft dit zelfs in zijn boek [4]. Helaas zorgde dit er wel voor dat mijn sprints niet klopte.

Een bijkomend nadeel is dat het domein van Norma vrij complex is, hierdoor heb ik ook veel tijd besteed aan het bespreken van het domein met Raymond.

#### Focus verleggen

Nadat ik een redelijk model had gecreëerd, besloot ik in samenspraak met mijn bedrijfsbegeleider om mijn focus te verleggen. De prioriteit om, in agile stijl, de user interface constant bij te houden werd achterwege gelaten. Ik heb mijn focus toen vooral gezet op de Rule engine integratie. Dit heeft uiteindelijk zijn vruchten afgeworpen, aangezien ik in minder dan twee weken betere conclusies kon trekken dan voorheen.

#### Voordelen

Deze nadelen hadden ook voordelen. Door het redelijk grondig onderzoek en gebruik van Domain Driven Design, heb ik nu een goed beeld wat het inhoud en manier om het toe te passen. Een bijkomend voordeel was dat de Rule engine integratie redelijk gemakkelijk was met het objectenmodel, dit bewees voor mij dat ik het objectenmodel op een degelijke wijze heb opgezet.

#### Veranderingen

Wat ik beter had kunnen doen, was beter gebruik maken van het evaluatie en analyse gedeelte in mijn sprints. In plaats van de sprints alleen maar te verlengen had ik het ook

kunnen aanpassen, waarbij ik de focus op de user interface sneller had kunnen verleggen. De tijd die aan het Domain Driven Design gedeelte is besteed lijkt mij onvermijdelijk omdat dat, na de Rule engine, het belangrijkste onderdeel was van het Proof of Concept. Wel had ik mijn doelen kunnen bijstellen en mijn planning beter kunnen aanpassen nadat ik merkte dat ik veel tijd kwijt was aan het maken van een Domain Driven Design.

# Deel III.

# Conclusie

# 8. Samenvattingen, Conclusies en Adviezen

In dit hoofdstuk worden een aantal conclusies besproken aan de hand van het uitgevoerde onderzoek. De onderzoeksfase en implementatiefase zullen kort worden samengevat. Ten slotte zal er een conclusie worden getrokken over de geschiktheid van een Rule engine voor Sogyo.

## 8.1. Samenvattingen

### 8.1.1. Samenvatting onderzoeksfase

De onderzoeksfase was opgesplitst in drie deelfases:

1. Vooronderzoek
2. Inventarisatie, eerste shortlist
3. Technische evaluatie, tweede shortlist

Het *vooronderzoek* behandelde wat een Rule engine was en gaf een beeld van de grote verschillen tussen de verschillende producten. De *iventarisatie* had als doel een betere blik te geven wat tegenwoordig beschikbaar is op de markt. Alsook te beoordelen welke producten op het eerste gezicht degelijk zijn. De *Technische evaluatie* ging wat dieper op de stof in. De producten werden op technische criteria beoordeeld, alsmede werd er een afweging gemaakt welke producten het beste bij de eisen van Sogyo passen.

#### Doel onderzoeksfase

Het doel van het onderzoek was een geschikt Rule engine product te vinden voor gebruik binnen Sogyo. De keuze voor een product is gemaakt aan de hand van de volgende factoren:

- Mens. Wie moet er mee om kunnen gaan? Wat vind een ontwikkelaar belangrijk?
- Proces. In welk proces moet het passen? Aan welke eisen moet het product voldoen om in het proces te passen?
- Tool. Hoe degelijk is de tool? Welke functionaliteit moet de Tool hebben? Wat is de leercuve van de tool?

**Eisen** Het product is gekozen m.b.v. verschillende eisen. Belangrijk waren de eisen vanuit Sogyo: het product moet met Java kunnen integreren, het moet in de huidige ontwikkelmethoden passen en ontwikkelaars vanuit Sogyo moeten ermee kunnen werken. Er waren ook technische eisen: Het product moet degelijk zijn en het product moet

voldoende functionaliteit hebben maar niet te veel. Aan de hand van deze eisen is uiteindelijk een product gekozen.

#### Uitgekozen Product

Drools is het uitgekozen product. Drools voldoet aan de eisen van Sogyo en is goed uit de technische evaluatie gekomen. De complexiteit en de leercurve kunnen helaas wel een nadeel zijn. Met Drools is dan ook het Proof of Concept uitgewerkt.

### 8.1.2. Samenvatting implementatiefase

In de implementatiefase werd het Proof of Concept ontwikkeld. Als project was het project Norma uitgekozen. Het Proof of Concept is opgesplitst in verschillende sprints. Deze sprints behandelde verschillende procesdelen van het Norma proces.

De motivatie om gebruik te maken van Sprints was om projectmethodieken te gebruiken van Sogyo. Binnen Sogyo wordt veel met agile projectmethodes gewerkt.

#### Doel implementatiefase

Het doel van de implementatiefase was om een Proof of Concept op te leveren. Dit Proof of Concept moet inzicht leveren op het gebruik van een Rule engine door Sogyo. Op basis van het Proof of Concept zijn er een aantal conclusies getrokken. De eisen voor het Proof of Concept zijn:

- Het Proof of Concept wordt op een DDD (Domain Driven Design) wijze opgezet.
- Het Proof of Concept zal gebruik maken van een bestaand project als casus.
- Het project dat uitgekozen wordt, moet zich lenen voor een vergelijking van de verschillen.

Deze eisen moeten waarborgen dat een goede vergelijking kan worden gedaan met het oorspronkelijke project evanals genoeg inzicht geven over het gebruik van een Rule engine om een advies te geven aan Sogyo.

Project Norma was uitgekozen als casus voor het Proof of Concept. Het domein van Norma is de basis voor het Proof of Concept.

#### Domain Driven Design

Domain Driven Design is een belangrijk element in het Proof of Concept. Het domein van Norma is met deze methodiek opgezet. Hiervoor is gekozen om de toepasbaarheid voor Sogyo beter te kunnen bewijzen, aangezien Sogyo veelvoudig gebruik maakt van deze methodiek. Er is vooral gebruik gemaakt van het Domain Driven Design boek van Eric Evans [4] om bepaalde Domain Driven Design technieken te implementeren in het Proof of Concept.

#### Rule engine

Drools is gebruikt voor de implementatie van een Rule engine in het Proof of Concept. Drools is vooral gebruikt voor de implementatie van business regels. Bij elke regel is een beslissingsproces vooraf gegaan over het gebruik van een Regel in de Rule engine. In het Proof of Concept zijn de volgende drie soorten regels terug te vinden:

- Validatieregels.
- Procesregels.
- Calculatieregels.

Dit zijn de drie soorten regels uit het domein waarvoor gekozen is om deze te implementeren in het Proof of Concept.

## 8.2. Conclusie

Om een conclusie te trekken over de toepasbaarheid van een Rule engine voor Sogyo, zullen er een tweetal deelvragen besproken worden:

1. Is een Rule engine toepasbaar voor gebruik binnen het Norma project?
2. Past een Rule engine in een Domain Driven Design project?

Deze vragen zijn cruciaal om een beeld te vormen over de toepasbaarheid van een Rule engine voor Sogyo. Op deze twee vragen is antwoord gekregen tijdens de implementatie van het Proof of Concept.

### 8.2.1. De toepasbaarheid van een Rule engine voor Norma

Wat het Proof of Concept heeft aangetoond is dat een Rule engine *niet* geschikt is voor gebruik in project Norma. Dit werd tijdens de Rule engine integratie duidelijk. Ik heb mijn bevindingen gedeeld met Raymond Tukkers (een ontwikkelaar van het orginele project) hij kon mijn bevindingen beamen. Deze conclusie is ook besproken met de bedrijfsbegeleider, deze heeft aangegeven het een gedegen conclusie te vinden.

Hieronder volgt de motivatie achter deze conclusie:

#### Het Norma Proces

Wederom zal naar het Norma proces gekeken moeten worden om de conclusie te onderbouwen. Zoals 7.1.5 en later ook in 7.2.3 is besproken, is dat het Norma proces centraal staat. Het belangrijkste wat het domeinmodel moet modelleren is dit proces; het verdelen van budgetten over de verschillende werken en dit weer verdelen over de verschillende performers. Tijdens de communicatie is opgemerkt dat er voornamelijk over dit proces gesproken werd. Omdat dit proces zo ingewikkeld is, is het zelfs uitgetekend om een goed overzicht te verkrijgen. Tijdens onze gesprekken kwamen business regels alleen ter sprake als deel van het proces.

**Norma specificatie** Deze trend is ook terug te zien in de Vuik II documentatie, in deze documentatie wordt het proces in uitermate gedetailleerd besproken. Hieruit blijkt ook wat voor belang gehecht wordt door de domein experts aan het centrale proces. De regels die hier in staan zijn te vergelijken met beslissing nodes in een flowchart. De beslissingen zijn wel degelijk deel van het proces, maar alleen als beslissing over welke volgende stappen ondernomen moeten worden. In vergelijking met proces nodes (imperatieve stappen) komen ze ook bijna niet voor.

**Flow model** Deze observatie is ook terug te vinden 7.2.3 als er gesproken wordt over Drools Flow. Tijdens het opstellen van regels werd duidelijk dat het domein niet goed met een Rule engine te modelleren was. Daarom werd er gegrepen naar een Flow model, dit Flow model probeert de kennis van een expert in een 'flowchart-achtig' wijze te modelleren. Deze flow modellen zijn veel meer op een proces gericht dan een Rule engine. Dit model voelde veel natuurlijker aan. Voornamelijk omdat een flowchart het domein natuurlijk kon modelleren.

### Opstellen van Regels

Tijdens het destilleren van het domein en het opstellen van regels werd duidelijk dat het opstellen van deze regels behoorlijke wat moeite kostte. Uiteindelijk kwamen er regels in het systeem die daar niet thuis hoorden. Deze observatie wordt beschreven in sectie 7.2.4, waar wordt gesproken over calculatie regels. Naar mijn mening zijn dit achteraf geen semantisch correcte regels.

De hoeveelheid ontdekte regels viel ook erg tegen, veel konden niet goed in de Rule engine gemodelleerd worden. Vaak ontstond de neiging om de beslissingen in de Rule flow te zetten. Uiteindelijk is dat niet gedaan omdat er voornamelijk van de Rule engine gebruik gemaakt moest worden. De rule flow moest alleen ondersteunend zijn. In sectie 7.2.4 worden Procesregels besproken, wederom komt het woord proces terug, deze regels zijn in nature alleen ondersteunend voor het proces.

De regels hadden ook niet veel invloed op elkaar, alles was goed te overzien. De regels waren nooit complex. Vaak waren het meer beslissingen die een procesdeel startte, bijv: "Als de betaalronde al eerder is uitgevoerd, dan worden de eerdere betalingen ook meegenomen". Dit resulteerde in platte sequentiële knowledge base, wat volgens de literatuur vermeden dient te worden [15].

### Domein model

Evans heeft in zijn Domain Driven Design boek een uitgesproken mening over het domein model. Hij meent dat dit samen met de domeinexperts ontdekt moet worden. Het domeinmodel moet de belangrijkste aspecten van het domein centraal stellen, deze aspecten moeten op een correcte wijze gemodelleerd kunnen worden zowel technisch als semantisch. Hierover is al gesproken in 3.5.3. Evans zegt:

> "The basic constraint of *MODEL-DRIVEN-DESIGN* - that the model supports an effective implementation and abstracts key domain knowldege"[4]

In mijn opinie is de “key domain knowledge” hier het proces, dus is het noodzakelijk deze op een effectieve manier te implementeren.

#### Samengevat

Een Rule engine is niet geschikt voor het Norma project omdat het domein zich niet goed laat transleren naar een Rule engine. De Rule engine pakt voor Norma de verkeerde problemen aan. Om het domein van Norma succesvol te modelleren moet nadruk gelegd worden op het centrale proces, dit proces moet duidelijk aanwezig zijn. De Rule engine is hier niet de juiste tool voor. Dit zou eventueel beter opgelost worden in een business flow model of zelfs in een conventioneel objecten model.

Als mij nog een keer dit project zou worden aangeboden voor een Rule engine integratie, zou ik het Norma project afwijzen op de grond van het proces gerichte domein en het gebrek aan complex business regels.

### 8.2.2. Domain Driven Design en Rule engines

Wat het Proof of Concept heeft aangetoond is dat een Rule engine een onderdeel kan zijn van een Domain Driven Design. Hieronder volgt de motivatie:

#### Theorie

Bij Domain Driven Design ligt de nadruk op het domein model. Dit model staat centraal in de applicatie. Het grootste gedeelte van de tijd is dit een objecten model, dit is echter niet noodzakelijk. Het domein kan ook op een andere wijze gemodelleerd worden. Een Domain Driven Design en Rule engines *kunnen* daarom samenwerken. Hier ligt de nadruk op het woord ‘kunnen’, de opzet van het project is namelijk belangrijk. Om Domain Driven Design en Rule engines te kunnen laten samenwerken moet aan de volgende eisen worden voldaan:

- De Rule engine is onderdeel van het domein model.
- Vanaf het begin af aan moet de Rule engine geïntegreerd worden in het project.

Het is belangrijk om de mindset te veranderen, er wordt gebruik gemaakt van een andere paradigma, namelijk een Rule engine. Het model blijft echter aanwezig, daarom is het belangrijk om de Rule engine deel van het domein model te maken.

Als een Rule engine en objectenmodel in een enkel project zitten dan moet er gebruik worden gemaakt van een enkel model, waar zowel het objectenmodel en de Rule engine hun eigen taak in hebben. Weggestapt moet worden van de gedachte van een los Domain Driven Design model en een Rule engine, de Rule engine is deel van het model. Het is noodzakelijk dat de Rule engine aan dezelfde regels voldoet als het eventuele objectenmodel. Daarom is het cruciaal dat een Rule engine het domein kan representeren, het model moet gevormd worden door het domein en de belangrijkste elementen, niet door de technologie die ervoor wordt gebruikt. Zodra dat gebeurd is het domeinmodel niet meer correct en is niet meer te spreken van een Model Driven Design.

Een model kan zelfs bestaan uit alleen een Rule engine en nog steeds een Domain Driven Design model hebben. Het model hanteert dan alleen een andere paradigma dan gebruikelijk. In het boek van Evans wordt veel gesproken over Patterns voor objectenmodellen, de kern van Domain Driven Design is echter bij meerdere pardigma's toe te passen. Zelf geeft Evans het volgende voorbeeld:

> "Although it has never reached the mass usage that object-oriented languages have, the Prolog language is a natural fit for *MODEL-DRIVEN-DESIGN*. In this case, the pardigm is logic, and the model is a set of logical rules and facts they operate on."[4]

Eric Evans haalt zelf ook het onderwerp Rule engines aan in zijn boek, het onderdeel waar de Rule engines worden aangehaald gaat over het gebruik van meerdere paradigma's in een enkel project. In dit geval spreekt hij over een objectenmodel en een Rule engine. Evans zegt dat gewaakt moet worden dat beide paradigma's onderdeel blijven van hetzelfde model. Het volgende citaat geeft zijn mening goed weer:

> "It is important to contine to think in terms of models while working with rules. The team has to find a single model that can work with both implementation paradigms. This is not easy, but it should be possible if the rules egine allows expressive implementation. Otherwise, the data and the rules become unconnected. The rules in the engine end up more like little programs than conceptual rules in the domain model. With tight, clear relationships between the rules and the objects, the meaning of both pieces is retained "

Het is noodzakelijk dat een Rule engine vanaf het begin onderdeel is van het project. Zo niet, bestaat de kans dat het model al gevormd is, de verantwoordelijkheid van de Rule engine is reeds overgenomen door het objectenmodel. Het is vervolgens onmogelijk om de integratie tussen het domein en de Rule engine naadloos te krijgen, het zullen dan twee aparte programma's worden.

Daarbij zegt Evans dat alleen door met de code te werken het model kan worden gevormd. Dit zelfde proces moet doorgemaakt worden door de Rule engine als deze goed het domein wil weerspiegelen. De Rule engine is immers ook onderdeel van het domeinmodel.

### Praktijk

De technische koppeling tussen de Rule engine en het objectenmodel was redelijk eenvoudig. Wat hiermee bedoeld wordt, is dat de technische integratie tussen het objectenmodel en de Rule engine redelijk eenvoudig was. Daarentegen was de semantische integratie tussen het objectenmodel en de Rule engine niet gemakkelijk, zoals in sectie 8.2.1 is te lezen. De technische integratie was gemakkelijk omdat het objectenmodel van een aantal Domain Driven Design patterns gebruik maakte. Dit is een bevestiging dat een objectenmodel en een Rule engine gecombineerd kunnen worden. Hieronder volgt een opsomming van een aantal patterns die de integratie vergemakkelijkten:

**Domein opsplitsen** Door het domein op te splitsen in Entities en Value Objects, wordt duidelijk welke feiten in de Rule engine mogen komen. Er is voor gekozen om alleen entiteiten als feiten in het systeem te injecteren. Immers is het alleen belangrijk om de state bij te houden van entiteiten. De taak van een Rule engine is state bijhouden en over de state van de objecten aan de hand van regels redeneren. Door het onderscheid in het objectenmodel, onstaat een duidelijke visie over welke objecten in de Rule engine mogen voorkomen.

**Gebruik van Factories** Voor de creatie van entiteiten, is gebruik gemaakt van factories. Evans geeft in zijn boek aan dat dit vaak gewenst is. Aangezien de constructor niet verantwoordelijk is voor logica die niet direct een correlatie heeft met de creatie van het desbetreffende object. De verantwoordelijkheid van Factories in het Proof of Concept is het injecteren van feiten in de Rule engine, in het Proof of Concept zijn dat entiteiten. Zo worden de constuctors van de entiteiten niet met deze verantwoordelijkheid belast en is snel terug te vinden welke entiteiten in de Rule engine geinjecteerd worden.

**Domain services** Door het gebruik maken van domain services, is direct duidelijk waar de Rule engine bootstrap en Rule engine specifieke functies thuis horen. De Rule engine is geëncapsuleerd in een facade genaamd *Drools*, die als een service fungeert. Deze facade regelde de injectie van verschillende objecten evanals het bijwerken van de objecten in de Rule engine. De Rule engine zelf is namelijk geen Entity of Value Object, een service is dus de perfecte interface naar de Rule engine.

Door de bovenstaande patterns, alsmede de losse koppeling tussen de domeinobjecten waardoor een hoop boilerplate code weggewerkt kan worden, was de technische integratie redelijk eenvoudig. Dit bewijst als het domeinmodel op een degelijke wijze is opgezet (dit wordt gewaarborgd door het gebruik van Domain Driven Design), de technische integratie van een Rule engine of andere pardigma's eenvoudig te verwezenlijken is.

### Samengevat

In mijn opinie kan een Rule engine gebruikt worden in een Domain Driven Design, mits er van te voren rekening gehouden wordt met de Rule engine.

Één van de grootste onzekerheden omtrent het gebruik van een Rule engine en Domain Driven Design, was dat de Regels uit het domein model worden gehaald. Dit moet inderdaad voorkomen worden, deze ontkoppeling zou resulteren in twee onafhankelijke programma's. De mindset moet veranderen wil een Rule engine succesvol worden gebruikt. Domain Driven Design en Rule engines staan niet los van elkaar, als een Rule engine gebruikt wordt moet deze onderdeel zijn van het design. Ofwel onderdeel zijn van het model die op een Domain Driven Design wijze is opgezet. De theorie die normaal gebruikt wordt voor het objectenmodel is vooral gericht op het domeinmodel, wat niet per se een objectemodel hoeft te zijn. Aangezien een Rule engine onderdeel is van het model is deze theorie net zo relevant voor de Rule engine.

Zodoende bestaat geen distinctie tussen Domain Driven Design en een Rule engine, de Rule engine is daar een onderdeel van.

### 8.2.3. De toepasbaarheid van een Rule engine voor Sogyo

#### Antwoord deelvragen

In de vorige twee secties zijn de volgende vragen beantwoord:

1. Is een Rule engine toepasbaar voor gebruik binnen het Norma project?
2. Past een Rule engine in een Domain Driven Design project?

Deze vragen zijn op de volgende wijze beantwoord:

1. Een Rule engine is niet toepasbaar voor gebruik binnen het Norma project. De oplossing is niet optimaal en lastig te realiseren. Een Rule engine geeft oplossingen voor problemen die voor Project Norma niet van belang zijn.
2. Een Rule engine kan gebruikt worden in een Domain Driven Design project, mits deze onderdeel is van het domeinmodel. Dit wil niet zeggen dat een Rule engine geschikt is voor elke Domain Driven Design project. Deze keuze is grotendeels afhankelijk van het domein zelf. Het domein moet op een logische en expressieve wijze gemoddeleerd worden, dit vereist gebruik van de juiste tools.

#### De toepasbaarheid

Op de vraag of een Rule engine toepasbaar is voor gebruik binnen Sogyo is het volgende antwoord gevonden:

> Ja, een Rule engine kan van toepassing zijn voor gebruik binnen Sogyo, mits de situatie er om vraagt.

Het Proof of Concept heeft aangetoond dat het gebruik van een Rule engine niet te verantwoorden is voor elk project. Deze conclusie kwam al aan de orde tijdens de onderzoeksfase, het Proof of Concept heeft dit bevestigd. Het Proof of Concept heeft aangetoond, dat een objectenmodel en een Rule engine op een Domain Driven Design wijze opgezet kan worden, als ervan uitgegaan wordt dat beide een andere aspecten belichamen van het domeinmodel. De Domain Driven Design ontwikkelmethodiek kan dus door Sogyo gehanteerd worden. Binnen Sogyo wordt flexibel omgegaan met het leren van nieuwe frameworks, tools, programmeertalen etc. Het is daarom aannemelijk dat het leren van een tool zoals Drools niet teveel moeite zal kosten.

De belangrijkste afweging is of een Rule engine juist is voor het model van een domein, dit zal niet in veel situaties het geval zijn. Een Rule engine kan in theorie gebruikt worden binnen Sogyo, het wordt ondersteund door de Domain Driven Design ontwikkelmethodiek, Sogyo heeft de vaardigheid om Drools te integreren in het ontwikkelproces. De toepasbaarheid is grotendeels afhankelijk van het domein van een project. Daarom moet goed afgewogen worden of een Rule engine voordeel biedt voor een project. Dit

voordeel bestaat alleen als het domein de mogelijkheid heeft om met een Rule engine het domein op een natuurlijkere wijze te kunnen weerspiegelen, dan alleen met een traditioneel objectenmodel.

### 8.2.4. Samenvatting

Sogyo heeft door dit document een beter inzicht gekregen omtrent het gebruik van Rule engines. Het Proof of Concept heeft antwoord gegeven over de toepasbaarheid van een Rule engine voor Sogyo. Een product is geadviseerd voor gebruik door Sogyo, namelijk JBoss Drools. De doelstelling van het project is door deze doelen te behalen.

## 8.3. Advies

Hieronder volgt een laatste advies voor Sogyo, omtrent het gebruik van een Rule engine in een project en voor een toekomstig project mocht er behoefte zijn aan een vervolgproject.

### 8.3.1. Project voor een Rule engine

In de scriptie is al meerdere malen herhaald dat het belangrijk is om goed af te wegen of een project geschikt is voor een Rule engine. Hieronder volgen een aantal vragen, deze kunnen gesteld worden aan het begin van het project, zodoende kan bepaald worden of een Rule engine geschikt is voor gebruik:

1. Bevat het project complexe business regels?
2. Zijn de business regels aan verandering onderhevig?
3. Hebben de regels invloed op meerdere aspecten van het domein?
4. Hebben de regels veel invloed op elkaar of zijn ze juist geïsoleerd?
5. Zijn de regels onoverzichtelijk, zijn de consequenties onduidelijk?
6. Zijn de regels voornamelijk voor validatie, of onderdeel van de kernfunctionaliteit?

Op minstens vier van de zes vragen moet positief worden geantwoord, voordat er überhaupt gekeken kan worden naar een Rule engine.

Ook zijn er een aantal eisen waar het project aan moet voldoen, wil er gebruik gemaakt worden van Drools of een andere Rule engine:

1. In het geval van Drools moeten de ontwikkelaars maken gebruik van Java (of eventueel C# voor Drools.Net).
2. De ontwikkelaars kunnen het veroorloven een extra tool te leren.
3. Er is toegang tot de kennis van domeinexperts.
4. De regels kunnen gedestilleerd worden uit het domein.

5. De Rule engine moet deel zijn van het domeinmodel.

6. De Rule engine moet vanaf het begin in het project gebruikt worden.

De belangrijkste afweging blijft of het model het toestaat om met een Rule engine gemodelleerd te worden. Daarom is het belangrijk, als er gekozen is voor een Rule engine om het eerst voor een klein deel van het domein te gebruiken. Zodoende kan er ontdekt worden wat er voor het model werkt, eventueel is hier wat refactoring voor nodig. Na deze test kan pas de daadwerkelijke beslissing worden gemaakt om een Rule engine te gebruiken.

### 8.3.2. Vervolgproject

Als er eventueel besloten wordt om een vervolgproject te starten kan het beste voor de volgende opzet gekozen worden:

> Kies een domein uit dat baat zou hebben bij het gebruik van een Rule engine. Indien mogelijk laat meerdere ontwikkelaars aan het project werken, zodat kan worden gezien hoe leesbaar de regels zijn als deze door meerdere ontwikkelaars gebruikt worden. Zorg dat de ontwikkelaars bekend zijn met de concepten van een Rule engine. Behandel de Rule engine als een onderdeel van het domein, onderwerp de regels aan dezelfde eisen die voor de rest van het model zijn gesteld. Zorg dat de Rule engine wordt gebruikt waarvoor deze bedoeld is: het bijhouden van business regels. De regels hoeven net zoals het objectenmodel niet in een keer perfect te zijn, refactoring is juist goed. Zorg dat het objectenmodel een natuurlijke integratie heeft met de Rule engine, hiervoor is waarschijnlijk een klein Domain Driven Design framework noodzakelijk.
>
> Probeer de regels net zoals het objectenmodel in meerdere iteraties op te bouwen. Zorg ook dat er test coverage is voor de regels evanals de facade naar de Rule engine. Na een aantal iteraties, ga nog eens na of de Rule engine daadwerkelijk geschikt was voor gebruik in het project. Blijf altijd in acht houden houden dat het model niet geforceerd met de Rule engine moet werken. Zorg dat dit moment niet te ver in de toekomst ligt zodat de Rule engine met wat refactorin vervangen kan worden. Tijdens het project moet misschien nog een performance slag plaatsvinden om de regels te optimaliseren.
>
> De volgende punten zijn belangrijk om in acht te houden te houden tijdens het opstellen van de regels:
>
> - Gebruik inferencing om duidelijke regels op te stellen.[1]
> - Als regels te sequentieel worden, ga dan na of het niet op een andere wijze kan.
> - Deel meerdere ‘if statements’ op in meerdere regels.

[1] Zie: `http://blog.athico.com/2010/01/drools-inference-and-truth-maintenance.html`

- Zorg dat de beslissingen altijd in in het predicaat worden genomen, nooit in de consequentie.
- Groepeer regels die een relatie met elkaar hebben.
- Herken het verschil tussen activatie en uitvoer, bedenk goed wanneer de Rule engines de regels moet uitvoeren.
- Zorg dat aanpassingen van objecten zoveel mogelijk via het objectenmodel gebeuren, laat eventueel de regels alleen functies van de entiteiten uitvoeren.

# 9. Evaluatie

In dit hoofdstuk wordt een korte evaluatie gegeven over het afstudeerproject. Over zowel de onderzoeksfase als de implementatiefase is in de scriptie al geëvalueerd, in dit hoofdstuk wordt nog een samenvatting gegeven van deze verschillende evaluties.

## 9.1. Fases

### 9.1.1. Onderzoeksfase

De onderzoeksfase verliep in zijn geheel zonder grote problemen. Allereerst is er een inventarisatie gedaan van de bestaande producten op de markt. Van deze producten zijn twee shortlists gemaakt, de eerste gebaseerd op algemene eisen, de tweede op de uitkomst van de technische tests. Ten slotte, is er een advies gegeven voor een product voor Sogyo. Het enige waar ik ben tegen aan ben gelopen is dat Openrules zo slecht presteerde. Gelukkig was de beslissing om dit product te laten vallen redelijk snel genomen. Het testen van producten duurde ook wat langer dan verwacht, door een strakke deadline te zetten is de planning hierdoor niet in gevaar gebracht.

Tijdens de implementatiefase liep ik tegen het probleem aan dat een aantal conclusies die ik had getrokken over Domain Driven Design niet geheel juist waren. De conclusies waren niet fout maar er zat nog wat meer achter. Ik merkte pas dat toen ik met Domain Driven Design ging werken, hoe cruciaal het is dat een Rule engine deel van het model moet zijn. Dit feit had gelukkig geen consequenties voor het gekozen product maar zorgde wel voor een beter inzicht over hoe Domain Driven Design en Rule engines gecombineerd konden worden. Ik heb achteraf mijn eerdere conclusie bekeken en aangepast indien noodzakelijk.

### 9.1.2. Implementatiefase

Ik ben tevreden met het resultaat geboekt in de implementatiefase. De implementatiefase was gericht op het implementeren van het Proof of Concept, dit Proof of Concept moest een aantal aspecten belichamen Tijdens de implementatie ben ik vooral tegen het feit aangelopen dat een goed Domain Driven Design minder makkelijk op te zetten was dan verwacht. Dit zorgde ervoor dat mijn aanvankelijke planning niet meer echt klopte. De nadruk lag vooral op de beslissingen die gericht waren op Domain Driven Design objectenmodel, de Rule engine integratie kwam minder aan bod. Daarbij moest de GUI regelmatig worden bijgehouden zoals bij Agile projecten het geval is. Op een gegeven moment is de beslissing genomen met mijn bedrijfsbegeleider om de nadruk vooral te leggen op de Rule engine integratie, na deze beslissing konden er veel sneller conclusies

worden getrokken. Achteraf denk ik dat de nadruk op Domain Driven Design, cruciaal is geweest. Het opstellen van het domeinmodel liet mij inzien dat Norma niet geschikt was voor gebruik i.c.m. een Rule engine. De lange tijd die besteed is aan Domain Driven Design had dus zowel goede als slechte punten.

### 9.1.3. Conclusie

Tijdens de implementatiefase werd duidelijk dat Norma niet geschikt is voor gebruik met een Rule engine. Dit was aanvankelijk wel vervelend, er zit namelijk veel tijd in het Proof of Concept. Echter was zonder het Proof of Concept totaal niet duidelijk geweest dat Norma niet geschikt zou zijn, de tijd is dus goed besteed. Het geeft zowel mij als Sogyo extra inzicht over het gebruik van Rule engines. Uiteindelijk heeft het Proof of Concept geresulteerd in een duidelijkere visie op zowel Domain Driven Design en Rule engine integratie. Achteraf ben ik dus tevreden met het resultaat.

## 9.2. Nawoord

Als allerlaatste wil ik nog vermelden, dat ik vind dat in zijn geheel een goed resultaat is afgeleverd. In dit afstudeerproject heb ik weer een aantal dingen geleerd, ik ben blij dat ik deze nieuwe kennis heb mogen vergaren. De opgedane kennis geeft mij een beter inzicht in software design en maakt mij uiteindelijk ook een betere programmeur.

# Bibliografie


[1] Greg Barton. Inferencing. mail, 2010.

[2] Jens Dietrich. Take google code page. http://code.google.com/p/take/, February 2010.

[3] Esper. Esper homepage. http://esper.codehaus.org/, February 2010.

[4] Eric Evans. *Domain-Driven Design, Tackling Complexity in the Heart of Sofware*. Addison-Wesley, 2006.

[5] Ernest Friedman-Hill. Jess homepage. http://www.jessrules.com, February 2010.

[6] IBM. Ilog jrules homepage. http://www-01.ibm.com/software/integration/business-rule-management/jrules/, February 2010.

[7] Innovations. Visual rules homepage. http://www.visual-rules.com/business-rules-management-software-rules-engine.html, February 2010.

[8] JBoss. Drools homepage. http://www.drools.org, February 2010.

[9] Alex Kozlenkov. Prova homepage. http://www.prova.ws/, February 2010.

[10] Norma. Over ons, norma. http://www.stichtingnorma.nl/index.php?15, April 2010.

[11] Openrules. Openrules homepage. http://www.openrules.com, February 2010.

[12] Mark Proctor. What is inference and how does it facilitate good rule design and maintenance. http://blog.athico.com/2009/11/what-is-inference-and-how-does-it.html, November 2009.

[13] Sogyo. Sogyo homepage. http://www.sogyo.nl, February 2010.

[14] Tibco. Tibco blogs. http://tibcoblogs.com/cep/2007/10/22/cep-vs-business-rules/, October 2007.

[15] Wikipedia. Chapter 1. the rule engine. http://downloads.jboss.com/drools/docs/5.0.1.26597.FINAL/drools-expert/html/ch01.htm, March 2010.

[16] Geoffrey Wiseman. Real-world rule engines. http://www.infoq.com/articles/Rule-Engines, June 2006.

# Appendices

# A. Interviews

## A.1. Interview met Raymond Tukkers

Op 15-02-2010 heb ik een interview gehouden met Raymond Tukkers die gewerkt heeft aan project Norma. Volgens mijn stagebegeleider Erik was dit een project met veel business rules in het domein en dus ook erg complex. Ik heb hem gevraagd of hij iets wist van DDD. Daar wist hij wel wat vanaf maar zag niet gelijk de tegenstrijdige belangen in DDD. Ook hebben we het gehad over project Norma dit project was inderdaad complex, maar dat zat meer in het ophalen van de data uit de database waar uiteindelijk een grote stored procedure voor is gebruikt.

Het leek hem niet heel erg handig om project Norma te pakken voor het uitwerken. Hij denkt dat dit te complex zou worden, project de leeuw zou misschien beter zijn.

De volgende punten kwamen er uit het gesprek:

- Dat de gebruiker zelf de business rules kan aanpassen moet nog maar blijken uit de praktijk.
- Dit zou inderdaad handig zijn om in een natuurlijke taal (DSL) te doen.
- Project de Leeuw eventueel beter dan project Norma.
- Eventueel is een Rule engine ook handig voor het scherp krijgen van de business rules, zodat de rules aangepast kunnen worden totdat er scherp is welke regels er nou echt zijn.

## A.2. Interview met Arno van den Uil

Op 16-02-2010 heb ik een interview gehouden met Arno van den Uil. Waar we over hebben gepraat waren de verschillende projecten bij Sogyo. Ook hebben we gepraat over welk project geschikt zou zijn voor gebruik met een Rule engine. Volgens Arno was project de Leeuw niet echt geschikt. In project de leeuw werken ze met een CATS methode, dit houdt in scores berekenen voor verschillende functies. Echter zijn dit niet echt regels maar meer het prikken in een matrix en een berekening uitvoeren. Er is ook niet echt een koppeling tussen regels. We hebben afgesproken dat we in maar nog terug komen op het project wat gekozen zou moeten worden.

Een Rule engine bij Sogyo zou niet gebruikt worden in hele grote projecten maar meer in wat kleinere MKB projecten dit zijn zijn namelijk de meeste projecten die Sogyo krijgt. De kans dat Sogyo iets met een Rule engine moeten maken voor een bank is vrij klein.

De volgende punten kwamen er uit het gesprek:

- Een groot product zal niet effectief zijn.
- Zorg dat het gemakkelijker is voor Developers dan voor gebruikers.
- Waarschijnlijk zullen gebruikers/experts niet in de praktijk de regels aanpassen.
- Project Norma waarschijnlijk beter dan project de Leeuw.
- Eventueel de regels uit Norma destilleren en in de Rule Engine zetten.

## A.3. Interview met Ralf Wolter

Op 19-02-2010 heb ik een interview gehouden met Ralf Wolter. We hebben gepraat over de basisprincipes van DDD (Domain Driven Design), maar ook hoe het wordt toegepast bij Sogyo. Ralf heeft uitgelegd hoe hij op een DDD manier projecten aanpakte d.m.v. technologieën zoals 'exploratory modeling'. Waar je samen met experts probeert het domein te modeleren in een zo'n normaal mogelijke taal. Hij verteld dat het domein ook vaak in een dynamische taal wordt gemodelled waar hij dan direct via de 'shell' tegen aan kan praten. Dit domein doet dan nog niks maar het raamwerk staat dan al, met zoveel mogelijk handelingen vertaalt van de werkelijkheid naar het domein.

Ralf vertelde ook dat het belangrijkste aspect van DDD encapsulatie is. Dit zorgt ervoor dat de complexiteit lokaal klein blijft. Ralf beweert dat de globale complexiteit wel groeit dat is onvermijdelijk maar de lokale complexiteit moet klein blijven. Uit methodes moet direct duidelijk worden wat het doet. Dus alles moet zo goed mogelijk genencapsuleert worden. Zo blijft de lokale complexiteit klein, om een deel van het programma te begrijpen zou je niet een heel ander deel van het programma te begrijpen. Als er bijv. een verhuizing wordt doorgevoerd hoeft het model geen kennis te hebben van de database.

Bij Sogyo wordt niet alles van DDD toegepast maar zeker wel 'exploratory modeling' gedeelte. Er zijn ook projecten geweest om dit te vergemakkelijken in statische talen zoals C# en Java.

De volgende kwamen er uit het gesprek:

- Bij DDD is encapsulatie erg belangrijk
- Testen en DDD gaan hand in hand.
- Bij Sogyo wordt DDD deels toegepast.
- Een rule engine zou in een DDD aanpak passen als het goed gencapsuleert wordt.
- Er moet dus voor gezorgd worden dat de lokale complexiteit met gebruik van een Rule engine klein blijft.
- Ook moet er rekening worden gehouden dat de Rule engine niet een programma apart wordt.
- Eventueel een idee om de Proof of concept op een DDD aanpak op te zetten.

# B. Productkeuze

Hieronder ga ik mijn productkeuze motiveren. Ik heb dus met algemene criteria een aantal punten toegekend aan verschillende producten. Wat ze allemaal gemeen hebben is wederom dat het Rule engines zijn die met Java kunnen integreren. De producten die uitgekozen zijn, zijn gekozen om hun score en/of compleet andere paradigma's.

De Rule engines die op een min getal uitkwamen (in tabel 6.2 alles lager dan nr. 8) heb ik verder niet naar gekeken; er is namelijk genoeg keuze. Hieronder zal ik de eerste acht producten toelichten en welke dan ook daadwerkelijk op de Shortlist staan.

## B.1. Producten

1. **Drools** (`http://www.jboss.org/drools`). Drools is een Production rule engine, die gebruikt maakt van 'forward chaining' en een aparte inference engine heeft. Drools is gebaseerd op een geoptimaliseerde versie van de Rete algoritme. Drools beschrijft zichzelf op de volgende wijze:

   > "Drools 5 introduces the Business Logic integration Platform which provides a unified and integrated platform for Rules, Workflow and Event Processing. It's been designed from the ground up so that each aspect is a first class citizen, with no compromises. " [8]

   Drools is gemaakt door JBoss en het is een community project. Hiermee wordt dus bedoelt dat het niet direct support ontvangt vanuit JBoss ofwel red hat. Hiervoor moet gebruik gemaakt worden van JBoss BRMS een 'sanitized' versie van Drools.

   Drools is in principe meer dan alleen een Rule Engine. Hiet biedt een heel platform aan. Dus de Rule engine kan makkelijk integreren met de 'event processor' en JBPM 'process/workflow orchestration engine'. Het voordeel hiervan is dus de gemakkelijke integratie. Maar ook dat de producten afzonderlijk van elkaar te gebruiken zijn.

   **Eerste indruk** Jboss Drools is op het eerste gezicht een modern en degelijk product. Het is goed gedocumenteerd en bevat van alle pakketten de meeste features. Het is mogelijk om via een web interface met JBoss Guvnor, bestaande rules aan te passen met een grafische tool die voor eindgebruikers ook gemakkelijk zou moeten zin. Ook is het mogelijk aan een bepaalde rule een stuk DSL te koppelen waarmee de parameters van de rule worden aangepast. Het biedt ook de mogelijkheid van een eclipse plugin voor het gemakkelijk ontwikkelen evenals een rule repository, dit is een soort versie beheer voor 'rules'. Rules kunnen worden gespecificeerd in de

Drools taal of eventueel via decision tables in Excel, ook is er zoals eerder gezegd ook een grafische 'rule tool' aanwezig.

Er zijn ook een aantal boeken aanwezig over de Drools omgeving dus dit zou het ermee leren werken ook gemakkelijker moeten maken. Ook is er een minder uitontwikkeld .net versie van Drools.

*Drools staat op de shortlist.* Het heeft de hoogste score van de open source producten omdat het op elk punt er goed scoort, het is nog steeds in ontwikkeling en wordt steeds uitgebreider in de toekomst staan ook features als 'backward-chaining' in de planning wat het mogelijk nog interessanter zal maken, ook biedt het bijna alles wat commercieel producten ook bieden. Het is open-source onder ASL, dus mag commercieel gebruikt worden, wat ook gunstig is en het wordt gesteund door Red Hat. Het ziet er dus uit als een degelijk en interessant product wat volwassen is maar waar nog steeds actieve ontwikkeling plaats vindt en die goed mee kan met de commerciele producten.

2. **Visual Rules** (`http://www.visual-rules.com`) Visual Rules is een 'proprietary' product. Er moet voor dit product flinke licensie kosten worden betaald. Wel is er een trial te downloaden die 30 dagen functioneert. Het bijna alle features die drools heeft behalve dan een web interface, maar deze staat nog op de planning. Ook heeft het product geen aparte inference engine. Het compileert de rules op een slimme wijze naar zowel Java of .Net code. Visual Rules beschrijft zichzelf op de volgende wijze:

   > "Visual Rules is a market-leading Business Rules Management (BRM) platform that supports both business and IT professionals during the entire rule lifecycle. This unique and collaborative platform enables efficient management of business rules and provides intuitive graphical tools." [7]

   **Eerste indruk** Visual Rules heeft een ander wijze van regels definiëren. Dit gebeurt puur visueel er is dus geen DSL waar de rules in gedefinieerd worden. Wel is er dus voor gezorgd dat de visuele editor (In ieder geval op het eerste gezicht) erg goed uitgewerkt is. Support wordt ook aangeboden en documentatie is en in overvloede. Ook interessant is dat ook al heeft Visual Rules geen inference engine wel ingebouwd support zit om een externe 'rule server' te hebben.

   *Visual Rules staat op de shortlist.* Het biedt een wat andere manier van ontwikkelen en is een redelijk groot product ook is het gelukt om een trial van 5 maanden te verkrijgen zodat uitgebreid getest kan worden. Wel moet worden nagedacht over de licentiekosten deze komen hier namelijk dus wel bij kijken, deze liggen ook redelijk hoog namelijk: per workstation 8,000,- en per server 7,500,- euro. Als partner zijn er eventueel OEM licenses verkrijgbaar.

3. **ILog JRules** (`http://jrules.ilog.com`) ILog JRules is het grootste product uit het lijstje en wordt gemaakt door IBM. Het is een van de eerste rule engines op de

markt en maakt wel gebruik van een inferencing engine. Het gebruikt zowel een geoptimaliseerde versie van het Rete algortime als andere door IBM zelf bedachte algortimes. JRules heeft ontzettend veel features en oogt in het eerste opzicht heel complex. Het is dan ook een erg groot product wat vooral door grote organisaties gebruikt wordt. JRules beschrijft zichzelf als volgt:

> "IBM WebSphere ILOG JRules provides functionality to build and deploy rule-based applications for Java, mainframe and SOA-based environments." [6]

**Eerste indruk** JRules scoort ook vrij goed. Het is een erg volwassen product, zelfs meer dan de Visual Rules of Drools. De documenatie is aanwezig maar redelijk moeilijk te vinden, ook is het wat onoverzichtelijk, er is wel voldoende om mee aan de slag te kunnen. JRules is een product die in grote organisaties gebruikt moet worden, dus daar moet aan gedacht worden. Ook zijn de licenstie kosten hoog: 1060,- euro per PVU (Processor Value Unit). Het product is ook minder actief in ontwikkeling, vernieuwing vind dus langzamer plaats. Ik heb het product als iets minder relevant heb laten scoren, de licentie kosten zijn hoog en het product is erg groot en complex.

*JRules staat niet op de shortlist, meerdere malen geen antwoord gehad van support over verlengde evaluatie omgeving*

4. **Jess** (`http://www.jessrules.com`). Jess is een Rule engine die eigenlijk een soort Java implementatie is van CLIPS[1] maar tegenwoordig nog wat meer. Het is een forward chaining Rule engine net zoals Drools. Evenals drools is het een production Rule engine en maakt ook gebruik van een aparte inference engine. Jess beschrijft zichzelf op de volgende wijze:

   > "Jess is a rule engine and scripting environment written entirely in Sun's Java language by Ernest Friedman-Hill at Sandia National Laboratories in Livermore, CA. Using Jess, you can build Java software that has the capacity to 'reason' using knowledge you supply in the form of declarative rules. Jess is small, light, and one of the fastest rule engines available. Its powerful scripting language gives you access to all of Java's APIs. Jess includes a full-featured development environment based on the award-winning Eclipse platform."[5]

   **Eerste indruk** Jess heeft net zoals Drools een IDE in eclipse om in te ontwikkelen. De rules worden geschreven in een Lisp, wat volgens de maker van Jess hele declaratieve code oplevert. Het heeft geen web interface en er is geen gebruikervriendelijke DSL, wel is dit volgens fora wel eerder succesvol gemaakt. Het is een volwassen product, het was een van de eerste Java Rule engines op de markt. Het

[1] `http://en.wikipedia.org/wiki/CLIPS`

wordt wel minder actief ontwikkeld dan andere Rule Engines, de laatste versie was 5 november 2008.

Er is redelijke veel documentatie en ook een boek. Het heeft dus minder features dan Drools, Visual Rules en JRules. Is ook een forward chaining engine en maakt evanals gebruik van het Rete algoritme. Het is niet opensource wel gratis gebruik voor research en eventueel een home edition voor kleine organisaties.

*Jess staat op de shortlist.* Het is een degelijk product is met veel materiaal om te starten. Het heeft weer een ander soort paradigma dan drools. Rules in Lisp zijn waarschijnlijk wat meer declaratief maar ook weer complexer echter is het wel interessant om deze mee te nemen op de shortlist.

5. **OpenRules**(`http://www.openrules.org`) OpenRules is ook een RBMS product. Het is niet helemaal duidelijk of het een inferencing engine gebruikt, maar het lijkt van niet. OpenRules is anders omdat het er vanuit gaat dat rules het beste via excel decision tables geïmplementeerd kunnen worden. OpenRules beschrijft zichzelf als volgt:

   > "OpenRules is a full-scale Business Rules Management System (BRMS) that provides a powerful while simple Open Source software for Rules-based Application Development. Using familiar tools such as MS Excel or Google Spreadsheets and Eclipse IDE, you may create complex decision support systems around an enterprise-class Business Rules Repository. OpenRules facilitates involvement of business people in the rules development and maintenance; it offers the power of BRMS without sacrificing simplicity."[11]

   **Eerste Indruk** Openrules ziet er degelijk uit. Het heeft redelijk wat documentatie maar lang niet zo veel als andere Rule Engines. Ook wordt de Excel functie ook aangeboden door andere Rule Engines. Het is GPL voor open source project maar een licentie is nodig voor commerciële projecten. Support is ook aanwezig. Intressant is dat het een rule solver en Machine Learning heeft geïntegreerd voor optioneel gebruik.

   *Openrules op de shortlist.* Openrules is een redelijk interessant product en kan eventueel worden meegenomen in de shortlist het biedt niet zoveel features als vooral grote commerciële spelers, maar biedt wel een andere manier van werken en wil volgens de website ook de simpelheid behouden. Er wordt betaald per server (dus installatie) dit kost per installatie 2200,- euro.

6. **Esper** (`http://esper.codehaus.org/`). Esper is geen traditionele Rule engine maar een CEP (Complex Event Processor). Dit houdt in dat het een tool is om events op te kunnen merken en daardoor acties uitvoeren. In verschillende bronnen waaronder een blog van een medewerker van Tibco[14]. Zou een CEP een rule engine kunnen vervangen, ik heb deze dus wel meegenomen in mijn eerste inventarisatie. Esper beschrijft zichzelf als volgt:

> Esper and NEsper enable rapid development of applications that process large volumes of incoming messages or events. Esper and NEsper filter and analyze events in various ways, and respond to conditions of interest in real-time. [3]

**Eerste indruk** Esper ziet er goed uit. Het is anders dan de producten uit de lijst maar desalniettemin is de documentatie duidelijk. Esper biedt ook support aan en er is een GPL versie en een commerciële versie. Het product bestaat al redelijk lang maar wordt nog steeds erg actief ontwikkeld

*Esper staat niet op de shortlist.* Ik heb na wat meer onderzoek toch niet gekozen dit product op de shortlist te zetten aangezien het niet helemaal duidelijk is of dit daadwerkelijk een vervanger is voor een Rule engine. Ook is de manier waarop er mee gewerkt moet worden volstrekt anders. Daarom heb ik ervoor gekozen deze niet mee te nemen. Voornamelijk om het onderzoek in te kaderen.

7. **Take** (`http://code.google.com/p/take/`) Take noemt zichzelf meer een Rule Compiler dan een rule engine. Het maakt niet gebruik van een inference engine. Take is een redelijk nieuw product en nog niet lang in ontwikkeling. Het is ook een erg klein en simpel product. Take zegt over zichzelf:

   > "TAKE consists of a scripting language that can be used to define derivation rules, and a compiler that creates executable Java code and deploys it into running systems. TAKE is inspired by Mandarax, has a similar API (and will have an adapter to import mandarax knowledge bases) but does not use a separate interpreter 'inference engine'. There is an Eclipse plugin that can be used to edit and compile TAKE rules."[2]

   **Eerste indruk** Take is nog een redelijk jong product. Vanaf 26 juli 2009 zijn ze bezig met een nieuwe versie van take. Deze is nog niet af. Ook zie ik sinds oktober geen veranderingen meer op de svn server. Het kan goed dat het nog een redelijk lange tijd duurt voordat de volgende versie mee komt. Interessant is wel dat dit een tegenhanger is van Visual Rules die ook de rules compileert naar java code, uiteraard heeft he voor de rest minder features.

   *Dit product staat niet op de shortlist. Andere producten bieden dit moment dezelfde functionaliteit en meer.*

8. **Prova** (`http://prova.ws`) Prova is een ander soort Rule engine. Om de regels te declareren wordt er gebruik gemaakt van Prolog. Prova maakt wel gebruik van een inference engine met chaining die gelijkenis vertoond aan die van Prolog. Het Prova project is afgeleid van de mandarax engine. Prova beschrijft zichzelf als volgt

   > "Prova is an economic and efficient open source rule language for creating distributed collaborating agents. Running in Java runtime,

> it combines Prolog style logic inference with extensions for Java scripting, concurrent and scalable reactive messaging, workflows and event processing suitable for Enterprise Service Bus deployment."[9]

**Eerste indruk** Prova is een degelijk product. Misschien nog niet helemaal klaar voor gebruik, maar er wordt wel veel aan ontwikkeld. Het is echter de vraag of deze rule engine ook toepasbaar is voor Sogyo. Het biedt wel een heel andere paradigma om in te programmeren dan de andere Rule engines in deze lijst. Ook bied het CEP mogelijkheden, maar dit is optioneel.

*Dit product staat niet op de shortlist. Er staat vier producten op de shortlist, dit product biedt ondanks het totaal andere paradigma niet genoeg om op de shortlist te komen*

9. **Termware** (`http://www.gradsoft.ua/products/termware_eng.html`) Termware is niet relevant. Het is geen geschikt product om te gebruiken als een 'Production rule engine' en noemt zichzelf dan ook een 'rule processing engine'. Dit product wordt dan uiteraard ook niet meegenomen op de shortlist.

### B.1.1. Keuze producten

Als bijsluiter wil ik er wel bij zeggen dat ik de hele grote producten expres uit mijn onderzoek heb gelaten. Van te voren was duidelijk dat grote Rule engines zoals Blaze advisor maar eigenlijk ook IBM JRules niet geschikt zouden zij voor Sogyo. Uit interviews met medewerkers werd duidelijk dat klanten niet groot geld zullen neer leggen voor de Rule engine.

Ook zijn die Rule engines meer gericht op bedrijven die het als het een core concept willen gebruiken in hun business model. Dit is in Sogyo ook niet het geval, dus om deze mee te nemen is bij voorbaat al niet nuttig voor Sogyo.

Ik heb er wel voor gekozen om Visual Rules, een commercieel product mee te nemen. Dit is ook een redelijk groot product maar bied een hele andere manier aan van ontwikkelen namelijk zonder een 'inferencing', dus meer op een flowchart-achtige wijze. De licentiekosten zijn redelijk prijzig maar het leek mij interessant om minsten een enkel groot commercieel product mee te nemen.

# C. Technische evaluatie

In deze paragraaf volgt een evaluatie van de producten uit de shortlist:

## C.1. Drools

**Algemene Informatie** Drools is een open source product van JBoss en heeft een Apache software foundation licentie. Drools maakt gebruik van een aparte DSL om de regels te declareren. Feiten zijn in Drools Java beans. Drools heeft een eigen IDE in de vorm van een eclipse plugin. Drools maakt gebruik van een 'inference engine', die een combinatie gebruikt van code interpreteren en compileren.

**Installatie** De installatie was vrij eenvoudig. Drools maakt gebruik van een Eclipse 'update site'. Met behupl van deze update site worden alle benodigdheden en de Drools IDE geïnstalleerd. Een probleem was dat debugger niet werkte, daarom heb ik de nieuwste build gepakt, deze werkte prima. Deze oplossing heb ik via de mailinglist gevonden.

**Tools** De IDE is redelijk goed, het werkte allemaal prima en na de kleine fix de debugger dus ook. De editor van Drools is snel, met features als: syntax highlighting en auto completion. Drools heeft niet extreem veel features maar voldoende om mee uit de voeten mee te komen. Een volledige debugger en een audit log zijn bijvorbeeld aanwezig. Als aparte download is er een BRMS[1] systeem dat 'Guvnor' heet, deze heb ik niet gebruikt. Wat een verbetering zou zijn is een geïntegreerde testomgeving dit is nu alleen mogelijk vanuit 'Guvnor'. Om de volledige mogelijkheden van de drools IDE te gebruiken een Drools project worden aangemaakt.

**Definiëren van regels** Het definiëren van regels wordt gedaan in de Drools DSL. Deze is gemaakt in ANTLR[2], ik heb redelijk wat met ANTLR gewerkt, ik vind het een fijne tool maar de fouten zijn onduidelijk. Hier is wat werk in gestoken met als gevolg dat dit beter is dan de standaard ANTLR foutafhandeling. De 'pattern matching' faciliteiten zijn erg goed. De taal heeft ook een aantal handige constructies die goed gedocumenteerd zijn.

**Java integratie** Java integratie is een speerpunt voor het Drools project. De interface is eenvoudig en goed gedocumenteerd. In de gedocumenteerd voorbeelden miste wel een aantal klassen en dergelijke. Dit was vervelend want deze moesten zelf achterhaald

[1] Business Rule Management System, een systeem waar de nadruk ligt op het beheren van regels
[2] ANTLR is een parser generator, zie `http://www.anltr.org/`

worden. Als dit eenmaal duidelijk is, wordt alles voor de rest voor de hand liggend. Ik vond de Java integratie van Drools de beste van de geteste producten.

**Ontwikkeling** Er is redelijk snel te ontwikkelen met de Drools engine. Een goed zicht op onderwerpen als 'inferencing' en conflicthantering van regels is belangrijk. Dit is niet helemaal uit de documentatie te halen, er zullen daarom waarschijnlijk nog wel wat onduidelijkheden blijven, een aantal concepten zijn namelijk niet intuïtief. Als er eenmaal een beter zicht is op activatie/uitvoeren van regels, wat agenda groepen zijn etc. Dan gaat alles een stuk sneller.

**Stabiliteit** De IDE is stabiel. Ik heb een enkele keer een Eclipse fout gehad, maar het werkte veelal probleemloos. Ik heb tijdens de ontwikkeling van het testproject geen bugs of dergelijke ontdekt.

**Complexiteit** De algemene complexiteit is middelmatig tot hoog. De tools en Java integratie zijn erg simpel. Daarentegen moet er wel een redelijke kennis zijn van een aantal concepten die door Rule engines gebruikt worden. Bijkomend is dat het niet eenvoudig is om in één keer optimale regels te definiëren. Wat je voor deze complexiteit terug krijgt is wel een krachtige oplossing. Waar regels compact en duidelijk kunnen worden gedefinieerd.

**Duidelijkheid documentatie** De documentatie van Drools is duidelijk opgezet. In mijn opinie zou er wel meer achterliggende theorie in mogen zitten, aangezien dit nodig is om een aantal voorbeelden te begrijpen. Een aantal stukken code zijn niet geheel compleet, dit was wel storend maar meestal binnen een paar minuten op te lossen, het gaat dan vooral om objecten die dan nog aangemaakt moeten worden. Er is wel een grote hoeveelheid voorbeelden waarvan de code ook te downloaden is. In het algemeen is de documentatie dus goed.

**Support** Ik heb een vraag gesteld op de mailing list van Drools. Na wat correspondentie had ik binnen een dag mijn antwoord. Deze mailing list, heb ik gemerkt, is redelijk actief en de projectleden van het Drools project kijken er ook regelmatig op.

**Snelheid** De Drools oplossing was langzamer dan de algoritmische. Hoe meer feiten in het systeem, hoe langzamer het werd. Het was gelukkig nog redelijk te overzien. Daarbij was mijn oplossing zeer waarschijnlijk niet optimaal. In vergelijking met met het volgende product, Visual Rules, was de Drools implementatie langzamer.

**Positieve en negatieve punten**
Positief:

- Makkelijke installatie.
- Complexe constructies mogelijk.

- Tools redelijk goed.
- Veel voorbeelden met broncode in documentatie.
- DSL is erg declaratief, pattern matching mogelijkheden erg goed.
- Volledig product, gratis en open source.

Negatief:

- Documentatie bevat spelfouten en wat fouten in de code.
- Kennis van onderliggende onderwerpen redelijk vereist.
- Geïntegreerd Testen alleen in Guvnor.
- Tools minder uitgewerkt dan Visual Rules.

### C.1.1. Conclusie

Drools is de eerste engine die getest is. De Drools implementatie is als nul meting gebruikt, als ik met andere producten tot een vergelijkbaar resultaat kwam, in een kortere tijd, dan was dat product in principe efficiënter. Ik vond Drools al in al prettig werken, ik kwam redelijk snel tot een 'stateless' resultaat, een 'stateful' implementatie was wat lastiger. De Java integratie was erg simpel en de Java kant van de engine was snel gemaakt.

Drools is dus een goede keuze als product. De Java kant kan opgezet worden is zoals je zelf wilt. Kennis is echter vereist van de achterliggende onderwerpen, om goed met het product om te kunnen gaan. Ook moet er goed nagedacht worden over het opzetten van de verschillende regels, het wordt anders snel inefficiënt.

## C.2. Visual Rules

**Algemene Informatie** Visual Rules biedt een ander paradigma dan Drools en andere Rule engines in het algemeen. Het heeft geen inference engine en ook de pattern matching faciliteiten zijn redelijk beperkt. Er wordt op een 'flowchart-achtige' wijze regels beschreven. Op deze wijze wordt ook automatisch conflict resolution opgelost, de programmeur beschrijf namelijk zelf welke regels er als eerst uitgevoerd moeten worden.

Deze flowcharts heten in Visual Rules 'rule-flows', 'rule-flows' kunnen aan elkaar gekoppeld worden of afzonderlijk van elkaar worden aangeroepen. 'Rule-flows' zijn altijd stateless, elke keer moeten de feiten opnieuw aangeboden worden aan Visual Rules.

De code voor het gebruiken van de 'rule-flows' wordt volledig gegenereerd.

**Installatie** De installatie is net zoals bij Drools gemakkelijk. Er is een update-site voor een Eclipse plug-in, die automatisch de installatie uitvoert. Een stand-alone versie is ook beschikbaar.

**Tools** De tools van Visual Rules zijn erg goed. Deze tools zijn dan ook de beste van de geteste producten. De interface is erg duidelijk en reageert snel. Om de volledige mogelijkheden van Visual Rules te gebruiken moet er een Visual Rules project worden aangemaakt. Dit project wordt duidelijk opgezet en vanuit de interface is het mogelijk

om direct code te genereren. Testen zit geïntegreerd in de IDE en er kan vanuit een 'rule-flow', makkelijk een testcase worden opgezet. In deze testcase kunnen vervolgens ook de mock objecten kunnen worden gegenereerd.

**Definiëren van regels** Het definiëren van regels wordt gedaan met een visuele tool. Hier kan een programmeur redelijk makkelijk mee overweg. Het is ook mogelijk om de technische specificaties uit te zetten. Zodat een domein expert eventueel alleen de flow en de regels op globale wijze kan beschrijven. Een programmeur kan dan later de technische details invullen. De pattern matching faciliteiten zijn redelijk krachtig maar minder krachtig als die van Drools of Jess. Door het gebrek aan inferencing wordt alles gemakkelijker om te overzien maar zorgt daarmee wel voor een minder krachtig systeem. Als gevolg dat de 'rule-flows' redelijk snel groot worden. Mijn Drools implementatie was vijftig regels, voor de Visual Rules implementatie had ik al drie redelijk grote 'rule-flows' nodig.

**Java integratie** De Java integratie is redelijk gemakkelijk. Het iets minder overzichtelijk dan dat van Drools, i.v.b. met gegenereerde code. Maar ik kwam er vrij snel uit.

**Ontwikkeling** Ontwikkelen gaat snel in Visual Rules. Het duurt niet lang om het programma onder de knie te krijgen. Ik was sneller met mijn Visual Rules engine klaar dan mijn Drools engine. Alles moet via de interface gemaakt worden, dus af en toe is het zoeken naar verschillende functies. De Java integratie was ook geen probleem, dit faciliteert de Visual Rules IDE ook erg goed. Ook is er een debugger waarmee de 'rule-flow' gedebugged kan worden, dit draagt evenals bij aan een lagere ontwikkeltijd.

**Stabiliteit** Ik heb geen problemen met de stabiliteit ontdekt. Zowel de IDE als de programmatuur functioneerde naar behoren.

**Complexiteit** De algemene complexiteit ligt laag, alles is makkelijk om te gebruiken en te overzien. Een aandachtspunt is wel de grootte van de 'rule-flows'. Deze kunnen snel erg groot worden. Wat daarentegen wel weer kan zorgen voor redelijk complexe situaties.

**Duidelijkheid documentatie** De documentatie is duidelijk. Helaas is er maar een enkele 'tutorial' en weinig voorbeelden. Het is voornamelijk een referentie.

**Support** Een forum is niet aanwezig. Ik heb kort contact gehad met de support van Visual rules omtrent licenties. Dit werd erg netjes behandelt en de correspondentie was er vlot.

**Snelheid** De Game of life engine die gemaakt was in Visual Rules, afgezien van de procedurele, het snelst. De snelheidwinst komt voornamelijk omdat er geen inferencing plaats vind.

**Positieve en negatieve punten**
Positief:

- Tools zijn goed en professioneel.
- Makkelijk om te begrijpen.
- Makkelijke installatie.
- Uitgebreide documentatie.
- Code generatie erg gemakkelijk en werkt niet tegen.

Negatief:

- Dure licentie.
- Minder krachtig.
- Grote hoeveelheden 'rule-flows' worden eventueel onoverzichtelijk.
- Geen forum/mailing list.
- Java integratie iets onduidelijker dan dat van Drools.

### C.2.1. Conclusie

Visual Rules was het tweede product wat ik heb getest. Het boodt een ander paradigma om mee te ontwikkelen en is prettig om mee te werken. Visual Rules heeft een totaal andere manier van werken dan Drools of Jess, er wordt minder gebruik gemaakt van traditionele technieken. Visual Rules bevat een stuk eerste orde logica, evenals andere technieken die ook in een andere Rule engine worden gevonden. Alleen mist het wel inferencing, de vraag is daarentegen wel wanneer inferencing noodzakelijk is. Visual Rules zou dus in een aantal situaties een zeer geschikt product kunnen zijn.

Het grootste nadeel zijn de kosten, deze zijn per werkstation al 8000,- euro. Een licentie voor de executie server is 7500,- euro per CPU.

## C.3. Jess

**Algemene informatie** Jess is in principe een Java implementatie van CLIPS. De Lisp-actige syntax zorgt voor een kleine en compacte taal. Jess werkt als enige van de geteste producten met een interpreter, het is hierdoor mogelijk om vanaf de command line regels en feiten te definiëren en uit te voeren. Jess maakt net zoals Drools gebruik van inferencing, ook is het mogelijk functies en dergelijke te specificeren. Jess is daarbij ook een redelijk volledige Lisp implementatie, veel van de Lisp functionaliteit kan gebruikt worden. In Jess bestaat er geen 'stateless' functionaliteit, alles is 'stateful'.

Jess is een commercieel product, het vereist een licentie om gebruikt te worden.

**Installatie** De bestanden van Jess komen in een zip, zowel de binaries als de eclipse plugin. Deze moet dan ook handmatig geïnstalleerd worden. Dit is wel wat meer werk dan een eclipse update site zoals andere producten aanbieden. Na de installatie moest Eclipse met de '-clean' parameter opgestart worden, anders was niet alle functionaliteit aanwezig.

**Tools** Jess heeft in alleen een editor in eclipse zitten. De editor is redelijk spartaans maar werkt prima. Jess heeft een REPL (Read Eval Print Loop), in de REPL kan in de Jess taal worden geschreven. Dit is handig tijdens ontwikkelen, er is gelijk resultaat. Daarbij heeft Jess een aantal functies om vanaf de REPL de engine te inspecteren. Helaas is er geen geïntegreerde debugger in Eclipse, er is daarentegen wel een debugger in de REPL.

**Definiëren van regels** Jess regels worden gedefinieerd in een Lisp-achtige taal. De taal is erg krachtig en heeft ook veel mogelijkheden. Zelf merkte ik alleen niet veel voordeel, als ik het vergeleek met bijv. de Drools taal. Het is ook niet eenvoudig om te leren, de meeste tijd bij de Jess engine heb ik besteedt aan het lezen van verschillende bronnen. De leercurve was bij de Jess engine, vooraal de taal, het hoogst.

**Java integratie** Jess heeft een Java API. Met deze API kunnen feiten en ook eventueel regels in de engine worden geïnjecteerd. Bijna alle functies zijn van de engine zijn vanuit Java aan te roepen. Ik vond het echter net wat minder makkelijk werken dan de Drools engine en ben dan ook nog een tijdje bezig geweest met het Java gedeelte.

**Ontwikkeling** De tijd die nodig is om Jess te leren en te gebruiken is hoger dan dat van andere producten. Het is daarentegen ook erg krachtig. Het voordeel is, dat Jess is voortontwikkeld op een sterke basis, het CLISP expertsysteem. Helaas kon ik, door gebrek aan tijd, niet hetzelfde resultaat bereiken als met Drools of Visual Rules. De conclusie is dat leercurve van Jess, aanzienelijk hoger ligt in vergelijking met andere producten.

**Stabiliteit** De Tools waren naar mijn inzicht stabiel, ik heb geen mankementen ontdekt.

**Complexiteit** Jess is wellicht de meest krachtige tool maar ook de meest complexe. Dit komt voornamelijk omdat de Jess taal niet makkelijk is om te leren. De Lisp-achtige syntax zal misschien ook afschrikken, daarbij is net zoals Drools een goede inzicht nodig over de theorie achter Rule engines.

**Duidelijkheid documentatie** De documentatie is goed gestructureerd en erg duidelijk. Er is voldoende informatie beschikbaar en het gaat ook nog wat meer over de achterliggende concepten dan de Drools documentatie. Ook heb ik een deel gelezen uit het boek, dit boek is, naar mijn mening, zowel een goede referentie als handleiding.

**Support** Ik heb geen daadwerkelijke problemen ondervonden. Tijdens de productevaluatie, heb ik de mailing list in de gaten gehouden en deze is redelijk actief.

**Snelheid** Omdat mijn implementatie niet gelijk was aan die van Visual Rules en Drools. Kan ik hier weinig over zeggen. Een onvolledige implementatie (met minder functionaliteit) presteert beter dan de Drools implementatie maar minder goed als de Visual Rules implementatie.

**Positieve en negatieve punten**
Positief:

- Erg krachtig.
- Redelijk goedkope licentie.
- Goede documentatie, waaronder een duidelijk geschreven boek.
- REPL is fijn om mee te ontwikkelen.

Negatief:

- Tools karig.
- Erg complex.
- Leercurve is hoger.
- Op het moment, minder actief ontwikkeld dan de andere producten.

### C.3.1. Conclusie

Jess is het derde Rule engine product die ik heb getest. Ik ben er positief over, het is een zeer krachtige oplossing. De REPL is erg fijn om mee te ontwikkelen omdat je snel dingen kan uittesten, ook zijn de faciliteiten voor het bekijken van de engine, vanuit de REPL, prettig. Helaas is Lisp niet de makkelijkste taal om te leren voor Java of C# programmeurs, ondanks dat Jess wel een aantal imperatieve constructies toevoegt. De taal zelf is de meest krachtige van de geteste producten, waarschijnlijk voegt dit voor Sogyo echter niet veel toe. Ik denk daarom niet, dat de extra complexiteit versus functionaliteit het waard is.

## C.4. Open Rules

Voor Open rules ga ik niet dezelfde structuur aanhouden als bij de voorgaande producten. Ik heb het product namelijk niet volledig kunnen testen. Ik kreeg het niet goed werkend. De problemen begonnen al bij de installatie, voor de installatie kreeg ik een aantal Jar's opgestuurd. Vervolgens moest ik wat bestanden vervangen met deze Jar's. Na dit gedaan te hebben, kreeg ik meerdere fouten te zien, na wat onderzoek bleek dat ik nog meer moest vervangen dan was aangegeven.

Nadat ik de installatie heb kunnen afronden, functioneerde de plugin niet naar behoren. Open Rules maakt gebruik van alleen maar Excel files. Deze zijn via een Eclipse plugin te bekijken in een Excel viewer maar niet op te slaan. Dit is volgens de support een bekende fout. Het opslaan moet extern met Excel gebeuren. Verder kreeg ik een Open Rules project niet aan de praat met een ander project als codebase. Na anderhalve dag,

was mijn eclipse workspace ook nog corrupt door Open Rules. Nadat ik de workspace weer had hersteld, heb ik besloten niet meer verder naar het product te kijken.

### C.4.1. Conclusie

Gezien de kwaliteit van andere producten en dat het werken met excel files ook mogelijk is in Drools en Visual Rules kan ik adviseren dit product niet te kiezen.