

Masterscriptie Informatiekunde

Specificatie van Strategieën voor Requirements Engineering

Jeroen Roelofs

Afstudeernummer: 63 IK

Einddatum: 12 november 2007

Begeleider: Stijn Hoppenbrouwers

Radboud Universiteit Nijmegen



Voorwoord

In dit document beschrijf ik mijn onderzoek in het kader van de afsluiting van de masterfase van mijn studie Informatiekunde aan de Radboud Universiteit te Nijmegen.

Het afstudeeronderzoek is uitgevoerd binnen het 'Nijmeegs Instituut voor Informatica en Informatiekunde' (NIII), dat onderdeel is van de 'Faculteit der Natuurwetenschappen, Wiskunde en Informatica' (FNWI). Dr. S.J.B.A. (Stijn) Hoppenbrouwers van de afdeling 'Information Retrieval and Information Systems' (IRIS) heeft dit onderzoek begeleid.

Mijn persoonlijke gegevens zijn:

Naam: Jeroen Roelofs

Studie: Informatiekunde

Studentnummer: 0313300

E-mail studie: jeroenroelofs@student.ru.nl

E-mail thuis: jeroenroelofs@chello.nl

Telefoonnummer: 06-15272654

Mijn afstudeergegevens zijn:

Afstudeernummer: 63 IK

Begeleider: dr. S.J.B.A. (Stijn) Hoppenbrouwers

Referent: dr. P. (Patrick) van Bommel

Tijdens het schrijven van deze scriptie heb ik uit diverse hoeken hulp gekregen. Allereerst wil ik mijn begeleider Stijn Hoppenbrouwers hartelijk bedanken voor de tijd die hij in dit afstudeeronderzoek heeft gestoken. Met zijn enthousiasme en vertrouwen is de totstandkoming van deze masterscriptie altijd erg plezierig geweest. Daarnaast wil ik Patrick van Bommel bedanken voor het op zich nemen van de taak van referent. Tenslotte gaat er ook een bedankje uit naar iedereen in mijn omgeving die op een of andere manier betrokken is geweest bij mijn afstuderen.

Rest mij om de lezer veel plezier te wensen bij het doornemen van dit document.

Arnhem/Nijmegen, november '07

Jeroen Roelofs

Samenvatting

In deze masterscriptie is het afstudeeronderzoek van Jeroen Roelofs beschreven, dat begeleid is door dr. S.J.B.A. (Stijn) Hoppenbrouwers van de onderzoeksafdeling IRIS van de Radboud Universiteit te Nijmegen. De aanleiding van dit onderzoek is een artikel van deze onderzoeksafdeling. Hierin wordt een eerste aanzet gegeven tot een raamwerk waarmee bepaalde methoden gespecificeerd kunnen worden. Hiermee bevinden we ons in het vakgebied van de 'method engineering'. Het doel van deze masterscriptie is om dit oorspronkelijke raamwerk te verfijnen. Dit doen we door het in de praktijk te gaan gebruiken om de strategieën van een concreet proces van requirements engineering te specificeren.

De eerste concrete stap om dit doel te verwezenlijken, is het operationaliseren van het vakgebied 'requirements engineering'. Iedereen heeft namelijk andere ideeën over wat nu de beste manier is om tot een gedegen requirements analyse te komen. We hebben er voor gekozen om requirements engineering op te vatten op de manier waarop het aan onze eigen universiteit wordt bedreven. De begeleider van dit onderzoek is daarom tevens informant, aangezien hij de cursus hierover doceert. Bij de beschrijving van deze operationalisatie hebben we ons gericht op de belangrijkste producten die dit concrete proces van requirements engineering oplevert. Achtereenvolgens zijn dit use cases, scenario's, domeinmodellen, business rules en terminologische definities. Deze producten vormen samen de basis om de wensen van belanghebbenden over een toekomstig informatiesysteem zo goed mogelijk weer te geven en vast te leggen.

Voordat we begonnen met het specificeren van de strategieën van dit proces, zijn we uitgegaan van een bestaand, provisorisch raamwerk. Dit raamwerk bestond vooral uit een model in een graphstructuur en een aantal stappen. Voor het maken van een requirements analyse zijn zulke stappen bijvoorbeeld het maken van een use case en het maken van een scenario. Tijdens de modelleersessies waarbij de specificatie opgebouwd werd, kwamen we er meerdere keren achter dat er het een en ander ontbrak aan het oorspronkelijke raamwerk. Hierdoor is het raamwerk van toen flink geëvolueerd en zo veel bruikbaar geworden. Zo gebruiken we nu onder andere een procesmodel in de taal YAWL in plaats van een graphstructuur. Daarnaast hebben we een conceptuele beschrijving in de vorm van een ORM-model toegevoegd. Tenslotte hebben we nog een aantal andere constructies ingevoerd bij de stappen van de strategieën.

Het resultaat is een flink aangepast raamwerk waar we uiteindelijk alles mee uit konden drukken wat we kwijt wilden over onze opvatting van requirements engineering. Zo hebben we na een flink aantal workshops een overzichtelijke en goed werkende specificatie in elkaar gezet. Deze specificatie is de meest complete en recente beschrijving van de methodische aspecten van de cursus 'requirements engineering' die er is. Voor alle producten staat beschreven welke strategieën je als requirements engineer zou kunnen volgen om ze te creëren. Deze specificatie zal aan de studenten van de cursus aangeboden worden. Het kan hen en andere geïnteresseerden helpen om requirements engineering op een goede manier te bedrijven.

Naast de papieren versie van de specificatie die in deze masterscriptie te vinden is, hebben we ook een interactieve, online applicatie opgezet. Hierbij hebben we gebruik gemaakt van een databasestructuur, wat ons geholpen heeft om de specificatie tot in de puntjes te laten kloppen. De online specificatie zelf is voor meer mensen toegankelijk en de techniek biedt ons enkele voordelen boven de papieren specificatie. Zo kunnen gebruikers online bijvoorbeeld zelf navigeren door meerdere niveaus van de activiteiten. We sluiten dit onderzoek af door middel van een literatuurstudie. Op deze manier hebben we onderzocht in hoeverre ons uiteindelijke raamwerk overeenkomt met de 'state of the art' uit de wereld van de 'method engineering'.

Deze literatuurstudie hebben we pas achteraf uitgevoerd, aangezien we op deze manier een raamwerk konden opbouwen dat precies aan onze eigen eisen voldeed. Het resultaat is een consistent en compleet geheel. De conclusie hiervan is dat veel raamwerken soortgelijke elementen bevatten als het onze, maar dat ons raamwerk en de (online) specificatie daarvan toch deels uniek en erg bruikbaar is.

Inhoudsopgave

Lijst van tabellen	6
Lijst van figuren.....	7
1. Inleiding.....	8
1.1. Verantwoording.....	8
1.2. Theoretisch kader	9
1.3. Methode	9
1.4. Leeswijzer	9
2. De operationalisatie	10
2.1. Het vakgebied 'Requirements engineering'	10
2.2. Scope van dit onderzoek.....	11
2.2.1 Use cases.....	12
2.2.2 Scenario's	13
2.2.3 Domeinmodellen	14
2.2.4 Business rules.....	15
2.2.5 Terminologische definities.....	16
3. Het raamwerk	16
3.1. Motivatie vorm van het raamwerk.....	17
3.2. Definitie van het raamwerk	19
3.2.1 Naam van een activiteit	19
3.2.2 Context van de strategie.....	20
3.2.3 Strategie	22
3.2.4 Conceptuele beschrijving.....	26
3.2.5 Iteraties	26
3.2.6 Een volledig opgebouwde activiteit.....	27
3.2.7 Meta-model	28
3.3. Plaatsing van het raamwerk via literatuurstudie.....	28
3.3.1 Van Bommel et al.....	29
3.3.2 Mirbel en Ralyté.....	30
3.3.3 Van de Weerd et al.	32
3.3.4 Overige raamwerken	33

4.	De specificatie.....	34
4.1.	Uitwerking volgens het raamwerk.....	34
	Een requirements analyse maken	34
	Een problem statement maken	35
	Een use case survey maken	36
	Een use case maken vanuit een use case survey.....	37
	Een basis course of events maken	38
	Een alternative path maken.....	39
	Een exception path maken	40
	Een scenario maken vanuit een use case	41
	Een scenario maken vanuit een use case survey.....	43
	Een use case maken vanuit een scenario	44
	Een domeinmodel maken vanuit een use case	45
	Relevante typeconcepten identificeren in een use case	46
	De feittypen maken	47
	Een domeinmodel maken vanuit een scenario	48
	Relevante typeconcepten identificeren in een scenario	49
	Een business rule maken	50
	Een terminologische definitie maken	51
	Een geïntegreerd domeinmodel maken	52
4.2.	Interactieve uitwerking middels een database	52
5.	Conclusie en toekomstig werk	55
6.	Lijst van afkortingen	57
7.	Bijlagen.....	58
7.1.	Databasestructuur van de online applicatie.....	58
	7.1.1. Tabel 'activity'	58
	7.1.2. Tabel 'activity_strategy'	58
	7.1.3. Tabel 'activity_notes'	58
7.2.	Programmatuur van de online applicatie	59
8.	Zakenregister	61
9.	Literatuur.....	62

Lijst van tabellen

Tabel 1: use case template	12
Tabel 2: business rules catalog	15
Tabel 3: grafische weergave van een <i>activity name</i>	20
Tabel 4: lijst met afkortingen van voorbeeldspecificatie	21
Tabel 5: grafische weergave van <i>steps</i> zonder <i>order</i>	22
Tabel 6: grafische weergave van <i>steps</i> met <i>order</i>	23
Tabel 7: grafische weergave van <i>steps</i> met dubbele <i>order</i>	23
Tabel 8: grafische weergave van <i>extended steps</i>	23
Tabel 9: grafische weergave van <i>steps</i> met <i>immediate order</i>	24
Tabel 10: samenvatting drie soorten van <i>order</i>	24
Tabel 11: grafische weergave van <i>steps</i> met verschillende soorten <i>order</i>	24
Tabel 12: grafische weergave van een <i>guided step</i>	25
Tabel 13: grafische weergave van een <i>composed step</i>	25
Tabel 14: grafische weergave volledig opgebouwde activity	27
Tabel 15: lijst van afkortingen	57
Tabel 16: tabel 'activity'	58
Tabel 17: tabel 'activity_strategy'	58
Tabel 18: tabel 'activity_notes'	58
Tabel 19: programmatuur online applicatie	60

Lijst van figuren

Figuur 1: grafische leeswijzer	9
Figuur 2: basic course of events	12
Figuur 3: use case survey.....	13
Figuur 4: use case diagram	13
Figuur 5: scenario	14
Figuur 6: domeinmodel	14
Figuur 7: grafische weergave <i>strategy context</i>	20
Figuur 8: grafische weergave <i>strategy context</i> met 'Void', stopconditie en afkorting	21
Figuur 9: grafische weergave <i>split</i> met <i>conditions</i> en <i>multiple tasks</i>	21
Figuur 10: grafische weergave soorten <i>splits en joins</i>	22
Figuur 11: grafische weergave <i>conceptual description</i>	26
Figuur 12: grafische weergave expliciete iteraties binnen <i>strategy context</i>	26
Figuur 13: meta-model raamwerk	28
Figuur 14: graphstructuur raamwerk Van Bommel et al.....	29
Figuur 15: <i>strategic guideline</i> raamwerk Mirbel en Ralyté	31
Figuur 16: proces-data-diagram raamwerk Van de Weerd et al.....	33
Figuur 17: voorpagina van online applicatie	53
Figuur 18: activiteit binnen online applicatie.....	54
Figuur 19: activiteit na volgen samengestelde stap binnen online applicatie	54
Figuur 20: locatiebalk online applicatie	55

1. Inleiding

Het onderwerp van deze afstudeerscriptie is: 'Specificatie van Strategieën voor Requirements Engineering'. Requirements engineering is een belangrijk deelproces van het ontwerpen van een (toekomstig) systeem, waarbij de eisen van gebruikers en andere belanghebbenden zorgvuldig geïdentificeerd en gedocumenteerd worden. Er zijn diverse strategieën denkbaar om dit proces tot een goed einde te brengen. Toch blijkt dit in de praktijk nog niet zo gemakkelijk te zijn. In deze masterscriptie onderzoeken we hoe we deze strategieën kunnen specificeren, en brengen we dit vervolgens in de praktijk bij een concreet voorbeeld.

De onderzoeksvraag is daarom:

Wat zijn de strategieën die we voor een concreet requirements engineering proces kunnen specificeren middels een eigen gedefinieerd raamwerk?

In de komende secties zullen we allereerst uiteenzetten waarom dit onderzoek relevant is. Vervolgens beschrijven we het theoretische kader waarbinnen deze afstudeeropdracht uitgevoerd wordt. Onder 'methode' zetten we uiteen via welke deelvragen we dit onderzoek aan willen pakken. Tenslotte beschrijven we onder 'leeswijzer' hoe dit document in elkaar steekt en hoe het gelezen dient te worden.

1.1. Verantwoording

Op universiteiten wordt er tegenwoordig steeds meer aandacht gegeven aan requirements engineering. Dat is terecht, aangezien een gedegen requirements analyse één van de belangrijkste succesfactoren is voor de bouw van een informatiesysteem. Elk systeem wordt gebouwd met een doel: hetgeen een systeem moet gaan doen voor de gebruikers. Wanneer je de eisen van alle belanghebbenden niet goed boven water krijgt of verkeerd documenteert, heb je aan het eind van de rit een groot probleem. Het belang van requirements engineering lijkt ons hiermee duidelijk.

Relatief gezien is het vakgebied van de requirements engineering echter onvolwassen. Hoewel men in de academische wereld steeds meer goede requirements engineers probeert op te leiden, is dit nog niet direct terug te zien in de praktijk. Soms worden requirements in het kader van kostenbesparingen helemaal niet gedocumenteerd. Erg vaak gaan er dingen mis en weten onervaren requirements engineers niet goed hoe ze te werk moeten gaan. Wat moet je bijvoorbeeld doen om een use case op te stellen? En hoe leid je uit een use case een domeinmodel af?

Door te specificeren wat de mogelijk te volgen strategieën zijn, proberen we om requirements engineers een concreet hulpmiddel te bieden bij hun werk. Belangrijk hierbij is om geen benauwend stappenplan te willen maken; de precieze invulling van sommige strategieën blijft open staan. Dit onderzoek is dus zeker nuttig voor onervaren requirements engineers, maar kan ook ervaren mensen in de praktijk bijstaan wanneer ze het overzicht even verliezen of vergelijkingsmateriaal nodig hebben om hun eigen processen aan te scherpen. In het beste geval helpen we middels deze specificatie het vakgebied 'requirements engineering' naar een hoger plan.

Daarnaast constateren we zelf dat dit onderzoek vrij nieuw is. Uit de literatuurstudie is gebleken dat er in het vakgebied van de 'method engineering' wel onderzoeken te vinden zijn die ook trachten om middels een raamwerk iets te specificeren. Wij zullen dit echter toepassen op een concreet proces van requirements engineering, waarbij we voor een groot deel ons eigen raamwerk aanhouden. Ons onderzoek is daarmee zeker een toevoeging op bestaande onderzoeken.

1.2. Theoretisch kader

Eenzijds ligt dit afstudeeronderzoek verankerd in het kennisgebied van de *method engineering*. Dit deel van de scriptie betreft namelijk de literatuurstudie en het ontwerpen van het raamwerk om de strategieën uiteindelijk in te specificeren. Binnen de method engineering ontwikkelt men ook methoden om strategieën concreet uit te werken. Literatuur die we hier eventueel over vinden dient als inspiratiebron en zal gebruikt worden om tegenover ons eigen onderzoek te plaatsen.

Anderzijds ligt dit afstudeeronderzoek verankerd in het kennisgebied van de *requirements engineering*. In concreto valt dit samen met de laatste deelvraag van deze scriptie, waarbij we het raamwerk ook echt gaan gebruiken om een concreet voorbeeld te specificeren. In de wetenschappen van de Informatica en de Informatiekunde speelt systeemontwikkeling een grote rol. Een project waarin een systeem wordt ontwikkeld bestaat uit verschillende fasen. We onderscheiden hierbij het analyseren, ontwerpen, coderen, testen en beheren van het systeem. Requirements engineering zelf valt bij deze opdeling onder de analyse.

1.3. Methode

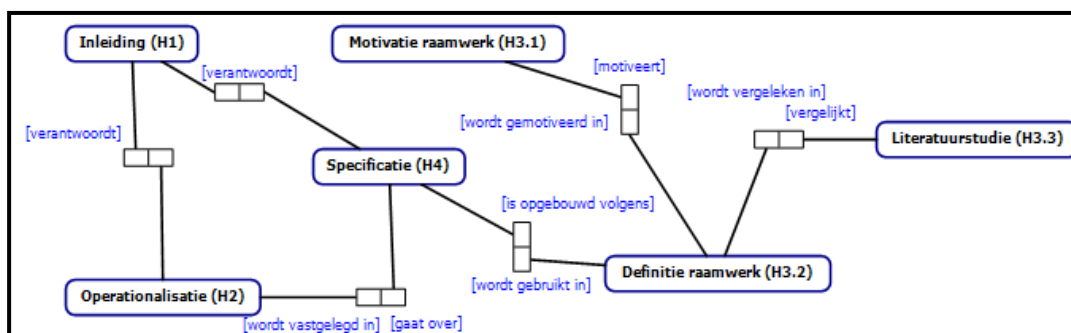
Om deze onderzoeksvraag adequaat te beantwoorden, splitsen we hem op in de volgende deelvragen:

- 1) Wat is een concreet voorbeeld waarmee we het begrip 'requirements engineering' kunnen operationaliseren?
- 2) Met wat voor een raamwerk kunnen we strategieën voor een dergelijk concreet voorbeeld specificeren?
- 3) Wat zijn de overeenkomsten en verschillen tussen ons onderzoek en andere reeds uitgevoerde onderzoeken waarbij er strategieën worden gespecificeerd middels een concreet raamwerk?
- 4) Wat is de specificatie van strategieën voor een concreet requirements engineering proces middels ons eigen gedefinieerde raamwerk?

De onderzoeksfunctie van dit gehele onderzoek is voornamelijk *beschrijvend*. We zullen een raamwerk beschrijven waarin we de strategieën kunnen specificeren, en zullen vervolgens een concreet voorbeeld van een proces van requirements engineering wederom proberen te beschrijven aan de hand van dit raamwerk.

1.4. Leeswijzer

Ten behoeve van het overzicht voor de lezer, presenteren we hieronder (zie figuur 1) een grafische weergave van de samenhang tussen de hoofdstukken in dit document.



Figuur 1: grafische leeswijzer

2. De operationalisatie

In deze scriptie gaan we gaan onderzoeken hoe we de strategieën van een concreet requirements engineering proces kunnen specificeren. Voordat we hier aan kunnen gaan beginnen, moeten we echter eerst duidelijk hebben wat we onder requirements engineering verstaan en wat daar bij komt kijken. Na een algemene beschouwing hierover, definiëren we de scope van dit onderzoeksgebied die relevant is voor ons onderzoek. Hieruit zal blijken dat we ons richten op de producten, want daar is het een requirements engineer uiteindelijk toch allemaal om te doen. Deze producten zullen we tenslotte kort beschrijven.

2.1. Het vakgebied 'Requirements engineering'

Een begripsanalyse is een essentieel onderdeel van een wetenschappelijk document. Het is bij het schrijven van een artikel van groot belang om eerst duidelijk uiteen te zetten wat je als auteur zijnde precies verstaat onder de begrippen die aan de orde komen. Hierdoor ontstaan er geen problemen doordat lezers verschillende interpretaties gebruiken. We zullen beginnen met een algemene bespreking van het vakgebied 'requirements engineering'.

In de wetenschappen van de Informatici en Informatiekunde speelt systeemontwikkeling een grote rol. Een project waarin een systeem wordt ontwikkeld bestaat uit verschillende fasen. We onderscheiden hierbij het analyseren, ontwerpen, coderen, testen en beheren van het systeem. Requirements engineering zelf valt bij deze opdeling voornamelijk onder de analyse. Het is één van de belangrijkste processen van de systeemanalyse.

Tijdens dit proces worden de eisen van gebruikers en andere belanghebbenden (de stakeholders) zorgvuldig geïdentificeerd en gedocumenteerd. Zo moet het uiteindelijke doel van het bouwen van het systeem boven water komen. Volgens Nuseibeh en Easterbrook is requirements engineering *"het ontdekken van dit doel, via het identificeren van belanghebbenden en hun behoeften, en het documenteren hiervan in een vorm die vatbaar is voor analyse, communicatie en implementatie"* (Nuseibeh & Easterbrook, 2000). Bij het identificeren van de behoeften dient de requirements engineer ook rekening te houden met de randvoorwaarden van het systeem. Zo kan het voorkomen dat de context vereist dat het systeem in de programmeertaal Java geprogrammeerd wordt. In ieder geval is dit een treffende definitie, die we in deze scriptie aan zullen houden.

De eisen die de stakeholders aan het systeem stellen noemen we requirements. Ze worden vaak gedefinieerd als meetbare uitspraken over de diensten die een systeem moet gaan bieden en de condities waaronder deze moeten worden uitgevoerd. Hierbij maken we onderscheid tussen functionele requirements en non-functionele requirements. Functionele requirements betreffen eisen over wat het systeem voor de gebruikers moet gaan doen. Non-functionele requirements gaan over de kwaliteit hiervan. Bijvoorbeeld hoe onderhoudbaar, betrouwbaar of gebruiksvriendelijk het systeem moet zijn. Het proces van requirements engineering bestaat uit drie deelprocessen: elicitatie, specificatie en validatie. Elk proces heeft zijn eigen uitdagingen en moeilijkheden.

Betrouwbare statistieken geven aan dat een groot aantal van zulke systeemontwikkelingsprojecten - waar requirements engineering deel van uitmaakt - niet succesvol eindigt. Jones heeft geconcludeerd dat hoe groter en complexer een systeem moet worden, hoe groter de kans is dat dit systeem er niet zal komen (Jones, 1996). Jones drukt de complexiteit uit in zogenaamde 'function points'. Om de gedachten te bepalen zullen we nu een indicatie geven van de orde van grootte waarin we moeten denken wanneer we een aantal functiepunten van een systeem horen.

Een voorbeeld van een systeem met meer dan duizend functiepunten, is een systeem met ongeveer 10-15 interfaces en 200 gebruikers. Bij tienduizend functiepunten moet je al denken aan rond de miljoen regels programmeercode. Als het te bouwen systeem nu honderd functiepunten heeft, is volgens Jones (1996) de kans dat het project voortijdig eindigt 6%. Betreft het een systeem van meer dan honderdduizend 'function points', dan is die kans al 80%.

In het gehele proces van requirements engineering kan er veel verkeerd gaan, wat Kulak en Guiney (Kulak & Guiney, 2000) uiteenzetten. Zo kan de requirements engineer een systeem gaan bouwen waar de opdrachtgever niet om gevraagd had. Ook kan hij grote fouten maken bij het programmeren of zelfs pas bij het testen ervan. Het is van groot belang op welk tijdstip zulke problemen ontdekt worden. Hoe later dit namelijk gebeurt, hoe duurder het oplossen ervan zal zijn. De kosten van wijzigingen na oplevering van een systeem zijn zestig tot honderd keer hoger dan de kosten van wijzigingen tijdens de systeemanalyse. Dat is een groot verschil. In de requirements engineering is het bekend dat veel problemen al ontstaan bij het boven water krijgen en opschrijven van de eisen die een bedrijf aan het systeem stelt. We kunnen hieruit concluderen dat het zeer belangrijk is om de systeemanalyse goed voor elkaar te hebben.

2.2. Scope van dit onderzoek

In de praktijk komt requirements engineering vooral neer op kunst in plaats van wetenschap, en het is moeilijk om te vatten hoe een goed proces er nu precies uitziet. Iedereen heeft daar zijn eigen ideeën over. Om niet te verdwalen in de doolhof van strategieën, zullen we het vakgebied van de requirements engineering verder moeten operationaliseren.

Met de beantwoording van deze eerste deelvraag blijven we dicht bij huis. Als concreet voorbeeld focussen we namelijk op de requirements engineering zoals die op de Radboud Universiteit – in de gelijknamige cursus – gedoceerd wordt. We beseffen dat we hiermee waarschijnlijk niet elke requirements engineer tevreden stellen. Dat is ook niet de intentie. Het gaat erom dat we een concreet voorbeeld kunnen specificeren van de manieren waarop dit proces bedreven wordt. Wij geloven zelf in dit proces, en de input van hoe dit proces eruit ziet bepalen we zelf aan de hand van onze ervaringen met de cursus 'requirements engineering'. Onze begeleider is daarbij tevens informant, aangezien hij de cursus doceert.

De nadruk van deze operationalisatie zal liggen op de producten waar het requirements engineering proces naar streeft. Het hoogste doel van een requirements engineer is namelijk het gedocumenteerd krijgen van de specificaties. Daarom richten we ons op de deliverables, en betrekken daar enkel procesmatige aspecten bij, als deze direct gericht zijn op de deliverables. Je zou bijvoorbeeld kunnen stellen dat een haalbaarheidsstudie ook bij requirements engineering hoort. Deze studie onderzoekt of de noodzakelijke voorwaarden in het bedrijf waar je bij werkt aanwezig zijn om het systeem überhaupt te kunnen realiseren. Dit proces is zeker belangrijk, maar leidt niet direct tot één van de producten die in de cursus 'requirements engineering' behandeld worden.

De primaire producten waar deze cursus wel op focust, zijn use cases, scenario's, domeinmodellen, business rules en terminologische definities. Deze producten samen vormen een compleet en samenhangend pakket dat requirements voldoende volledig weet te beschrijven. In de cursus van de Radboud Universiteit wordt intensief gebruik gemaakt van het boek 'Use cases, requirements in context' van Daryl Kulak en Eamonn Guiney. In dit boek (Kulak & Guiney, 2000) wordt een methode aangeboden om requirements op te schrijven, die gedreven is door use cases. Zoals gezegd is de methode van de cursus echter breder dan alleen use cases. We zullen alle genoemde deliverables – beginnend met use cases – in de komende vijf secties één voor één beschrijven.

2.2.1 Use cases

De zogenaamde ‘use cases’ zijn het centrale element van de verzameling deliverables waar we ons in deze afstudeerscriptie op richten. Ze stammen oorspronkelijk uit de UML-methodologie. Ze bestaan in essentie uit een opeenvolging van interacties binnen taken die actoren uit moeten voeren, al dan niet met het toekomstige systeem. Om het allemaal wat duidelijker te laten worden, gebruiken we voorbeelden uit een requirementsdocument betreffende een roostergenerator.

In een use case zet je onder de ‘basis course of events’ de stappen die normaal gesproken tot het takenpakket van een actor horen (zie figuur 2). Je zou dit kunnen zien als een verzameling van feitzinnen voor een specifiek deel van een systeem. Feitzinnen bestaan vaak uit een subject, werkwoord en bijbehorend predicaat. Ze zijn niet samengesteld, wat ze simpel en duidelijk maakt. De grote kracht van use cases is dat ze hierdoor door feitelijk elke stakeholders begrepen kunnen worden. Het is niet nodig om de syntax en semantiek van een modelleertaal of iets dergelijks te begrijpen. Daarmee is het helemaal geschreven in de taal van het bedrijf.

Basic course of events:

1. Docent neemt contact op met de studietoördinator.
2. Docent geeft de vakcode op.
3. Docent geeft de datum aan.
4. Docent geeft de begintijd van het vak aan.
5. Docent geeft de eindtijd van het vak aan.
6. Studietoördinator laat weten of de wijziging doorgevoerd kan worden.

Figuur 2: basic course of events

De definities van wat je precies wel en niet tot een use case dient te rekenen verschillen nogal. In de methode van de cursus ‘requirements engineering’ wordt met een use case meer bedoeld dan enkel deze ‘basic course of events’. In de volgende tabel staan de overige onderdelen van een mogelijke use case ingevuld, evenals een uitleg van deze onderdelen.

Onderdeel	Beschrijving
Use Case Name	Een unieke en duidelijke naam die de essentie goed weergeeft
Iteration	Fase van de use case, uitgedrukt in resp. ‘Facade’, ‘Filled’ of ‘Focused’
Summary	In enkele regels samengevat wat de essentie van de use case is
Basis Course of Events	Het meest voorkomende pad dat de actor met het systeem doorloopt
Alternative Paths	Eén of meerdere correcte paden die minder vaak voorkomen
Exception Paths	De paden die enkel voorkomen wanneer er iets mis gaat in het proces
Triggers	De reden waarom of wanneer je een use case in dient te gaan
Assumptions	De aannames waar je van uit gaat dat ze gelden tijdens de use case
Preconditions	Conditie van buiten de use case die moeten gelden om te beginnen
Postconditions	Conditie van buiten de use case die moeten gelden na afloop
Related Business rule(s)	Verwijzingen naar de business rules die van kracht zijn bij deze use case
Author	De auteur(s) van de use case, minst relevant en staat daarom onderaan
Date	De datum waarop de drie fasen (zie iteration) afgerond zijn

Tabel 1: use case template

Naast de al eerder beschreven ‘basis course of events’ vormen de ‘alternative paths’ en de ‘exception paths’ het hart van de use case. Het is altijd belangrijk om ook rekening te houden met interacties tussen de actor en het systeem die minder vaak voorkomen, of die juist voorkomen wanneer er zich een error voordoet. Deze stappen worden nog wel eens over het hoofd gezien. De overige onderdelen van deze mal zijn ook voor het bedrijf van belang. Zij moeten rekening houden met bijvoorbeeld pre- en postconditions in hun eigen bedrijfsprocessen.

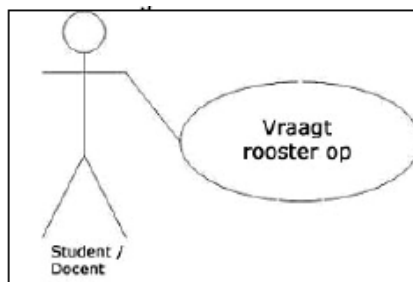
Deze methode van requirements engineering maakt ook onderscheid in een drietal stadia die een use case doorloopt. De eerste fase is de 'Facade', waarin de template nog niet helemaal ingevuld is, maar wel alle processen zo veel mogelijk afgedekt zijn. De tweede fase is de 'Filled', waarin het template helemaal gevuld wordt, en use cases ook typisch opgesplitst worden in uniekere delen. De laatste fase is vervolgens de 'Focused', waarin elke use case nog eens zeer kritisch bekeken wordt, waarbij overbodigheden geschrapt worden. Het principe is hier dus om eerst een beginnetje te maken, dan zo veel mogelijk mogelijkheden te selecteren en vervolgens deze verzameling zo ver mogelijk in te perken en te combineren tot je alleen de écht belangrijke elementen over hebt.

Vaak wordt er ook nog een zogenaamde 'use case survey' bijgehouden, waarin een overzicht van alle uiteindelijke use cases gegeven wordt. Hierin worden enkel de naam, actor, samenvatting en bron opgesomd. Daarnaast is voor elke use case aangegeven wat de complexiteit en prioriteiten er van zijn. In figuur 3 is een voorbeeld van één use case uit zo'n survey te zien.

Use Case Number	004
Use Case Name	Docent geeft wijziging aan.
Initiating Actor	Docent
Description	De docent wil een wijziging doorvoeren en moet hiervoor zijn wijziging doorgeven aan de betreffende studiecoördinator.
Completeness	Focused
Use Case Complexity	Hoog
Architectural Priority	Hoog
Business Priority	Hoog
Dependency	Gemiddeld
Source	Docent
Comments	-

Figuur 3: use case survey

Tenslotte wordt bij een use case ook vaak een zogenaamd 'use case diagram' toegevoegd. Dit diagram (zie figuur 4) bevat meestal niets meer dan de actor(en) van de use case en de bijbehorende taak. Het diagram blinkt niet uit in figuratieve mogelijkheden, maar toch blijken mensen het vaak fijn te vinden om de essentie van een use case in een andere vorm dan tekst te zien. Het is ook mogelijk om één groot use case diagram op te stellen om de samenhang tussen meerdere use cases te zien.



Figuur 4: use case diagram

Hoewel de use case het centrale element van de methode van 'requirements engineering' is waar wij hier op focussen, zijn de scenario's, domeinmodellen en business rules ook belangrijk en onmisbaar. We zullen ze in de volgende secties beschrijven.

2.2.2 Scenario's

Zoals gezegd vormt de 'basis course of events' samen met de 'alternative paths' en de 'exception paths' de kern van een use case. Deze paden bestaan echter wel alleen uit subjecten en predicaten op typeniveau. Zo is in de genoemde voorbeelden het subject vaak 'student' of 'docent'. In de werkelijkheid heb je niet te maken met het type 'docent', maar met een specifieke docent die luistert naar een specifieke naam, bijvoorbeeld 'Jasper van Rijn'.

Ook wordt er bij use cases gesproken over het type 'datum', en niet over de instantie '15 januari 2007'. Veel mensen hebben moeite met abstracties, en kunnen veel beter overweg met beschrijvingen op instantieniveau. Dat is de reden waarom er scenario's bestaan. Scenario's zijn namelijk op instantieniveau ingevulde paden uit zowel de 'basis course of events' als de 'alternative paths' en de 'exception paths'. Het zijn concrete voorbeelden van de processen, waar elke klant zich iets bij voor kan stellen. Zie figuur 4.

Scenario D: Docent Sjaak aan de RU Nijmegen

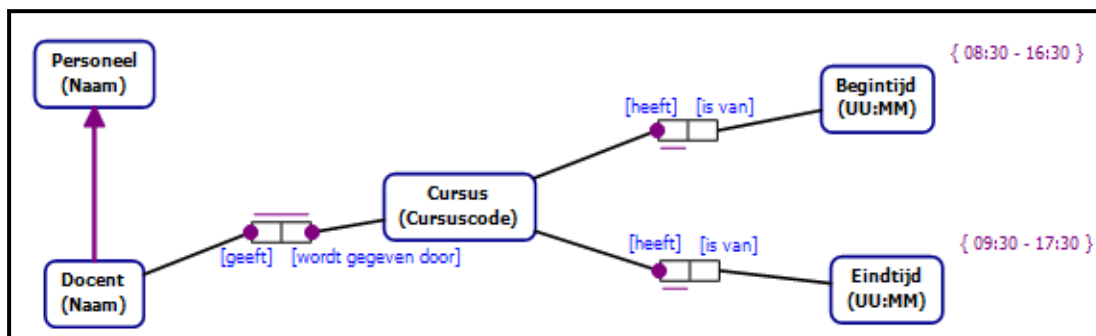
1. Docent Sjaak wil zijn vak iets later laten beginnen en eindigen.
2. Sjaak belt naar studiecoördinator Vera om de wijziging door te geven.
3. Vera neemt op en Sjaak geeft de nieuwe begintijd 10:30 op.
4. Sjaak geeft ook de nieuwe eindtijd 12:30 op.
5. Vera geeft als antwoord dat er op die tijden een tentamen wordt gegeven en dat de wijziging niet mogelijk is.

Figuur 5: scenario

Het is belangrijk om te benadrukken dat de verhouding van een use case ten opzichte van een scenario die van 1:n is. Dat betekent dat er bij een use case meerdere scenario's opgesteld kunnen worden. Het doel is om de scenario's op een dusdanige manier te kiezen, dat alle paden van de use case worden afgedekt. Het ligt voor de hand dat er dan scenario's ontstaan uit zowel de 'basis course of events' als de 'alternative paths' en de 'exception paths'. Maar daarnaast is het ook mogelijk en zeer gewenst dat er meerdere scenario's voortkomen uit slechts de 'basis course of events'. Het gaat er immers om dat de stakeholders de beschreven processen goed kunnen begrijpen.

2.2.3 Domeinmodellen

Een domeinmodel is een grafische weergave van de concepten en relaties die in een domein voorkomen. Hierdoor krijg je in één model een totaaloverzicht van de samenhangende delen te zien. Daarnaast is het gebruikelijk om hier via constraints beperkingen bij aan te geven.



Figuur 6: domeinmodel

Hierboven zie je een voorbeeld, waarin de concepten en relaties uit figuur 4 in terugkomen. Dit model is in het programma NORMA in de taal ORM (Halpin, 2001) geschreven. Deze afkorting staat voor Object Role Modelling, en ORM is in staat om zogenaamde feitzinnen in natuurlijke taal op een voor gebruikers intuïtieve manier weer te geven. Hoewel we in dit document gebruik zullen maken van ORM, is deze taal voor domeinmodellen allermintst heilig. Andere tools en modelleertalen zoals UML class diagrams (Booch, Rumbaugh, & Jacobson, 2005) zijn bijvoorbeeld ook mogelijk.

Een domeinmodel kan op verschillende manieren gebruikt worden binnen een specificatie van requirements engineering. Allereerst is het zeer nuttig om voor elke use case een apart domeinmodel op te stellen. In de 'basis course of events', 'alternative paths' en 'exception paths' komen nogal wat objecten en relaties voor.

Door middel van een domeinmodel kunnen deze precies worden uitgewerkt. Ook is het wellicht nuttig om naast de aparte domeinmodellen nog één groot, integraal domeinmodel op te stellen. Dit noemen we een 'generiek domeinmodel'. Hiermee herhaal je wel veel informatie uit de aparte domeinmodellen, maar zorg je wel voor meer overzicht en inzicht in de samenhang tussen alle aparte domeinmodellen.

Tenslotte kan het ook nog handig zijn om een zogenaamde 'domeinmodel populatie' in te vullen. Domeinmodellen beschrijven concepten en relaties voornamelijk op typeniveau. Dat wil zeggen dat we het over concepten als 'docent' hebben, in plaats van over instanties als 'docent Hoppenbrouwers'. Net als dat het nuttig is om interacties in een use case tevens weer te geven in scenario's, is het ook nuttig om een domeinmodel te populeren. Stakeholders die talen als ORM en UML niet begrijpen, begrijpen een dergelijke populatie wellicht een stuk beter.

2.2.4 Business rules

De bedrijfsregels zijn de geschreven en ongeschreven regels die voorschrijven hoe een bedrijf haar zaken zou moeten bedrijven. In essentie zou je ook kunnen zeggen dat business rules die regels zijn die niet in een domeinmodel te modelleren zijn. Wanneer er in een bepaald domein nog geen business rules expliciet gemaakt zijn, dien je ze als requirements engineer gewoon zelf op te stellen.

Bedrijfsregels nemen tegenwoordig een steeds grotere plaats in de specificatie van requirements engineering proces in. Het creëren van business rules is samengegroeid met het vakgebied van de domeinmodellering. Ze worden daarom op basis van domeinmodellen opgebouwd. Business rules dienen relevant voor het domein te zijn. Zo leveren ze ook een nuttige bijdrage aan bijvoorbeeld een use case.

Met betrekking tot het voorbeeld van de roosterviewer zou je bijvoorbeeld de volgende business rules als in tabel 2 kunnen opstellen. Een dergelijke opsomming van regels noemen we een 'business rules catalog'.

	Regeldefinitie	Bron
001	Beheerder verstrekt persoonsgegevens van studenten alleen als dit is voor studieondersteuning of andere doelstellingen van gegevensverzameling.	Vademecum: Reglement Bescherming Persoonsgegevens Studenten.
002	Tijdens de tentamenperiode van een cursus heeft de student voor deze cursus "collegevrij".	Studiegids NIII (website)
003	De doelen van een cursus, d.w.z. de competenties die geslaagde deelnemers na afloop beheersen, worden in de cursusbeschrijving vermeld.	Studiegids NIII (website)

Tabel 2: business rules catalog

Zoals af te leiden is uit bovenstaande tabel 2, is de vorm van business rules inderdaad gerelateerd aan een domeinmodel. In de eerste regel bijvoorbeeld kun je zo een aantal concepten (beheerder, persoonsgegevens, studenten) en een aantal relaties (verstrekken, zijn van) ontdekken. Deze concepten en relaties dienen ook expliciet voor te komen in het domeinmodel dat bij de desbetreffende use case hoort.

Over de vorm van een business rule valt echter meer te zeggen. In ons voorbeeld is het een soort natuurlijke taal, waarin we wel zoveel mogelijk gebruik maken van de exacte termen uit het bijbehorende domeinmodel.

Er zijn echter diverse talen in ontwikkeling waarmee bedrijfsregels nauwkeuriger opgeschreven kunnen worden. Eén daarvan is de taal ORC (Hoppenbrouwers, Proper, & Van der Weide, 2005), waarbij de afkorting staat voor Object Role Calculus. Dit is een semi-natuurlijke taal, wat wil zeggen dat het wel degelijk een vaste, logische structuur heeft, maar wel begrijpelijk blijft voor niet technische domeinexperts. De taal ORC is een geëvolueerde variant van de taal RIDL (Meersman, 1982). Hier bestaan weer twee varianten van, namelijk het meer op multisets gerichte LISA-D (Ter Hofstede, Proper, & Van der Weide, 1993) en het meer op de praktijk gerichte QonQuer (Proper, 1994). ORC heeft als doel om LISA-D en ConQuer te herintegreren.

Tenslotte speelt bij business rules ook het verschil tussen ‘alethic’ en ‘deontic’ een rol. Regels en constraints beschouwen we als ‘alethic’ wanneer het noodzakelijk is en absoluut niet geschonden kan worden. We spreken over ‘deontic’ wanneer de regel in principe wel overtreden kan worden. De regel dat een geleend boek op tijd teruggebracht moet worden naar een bibliotheek is een voorbeeld van een ‘deontic rule’. In principe is het namelijk mogelijk om het boek later terug te brengen, waarvoor de kosten dan voor eigen rekening zijn. De regel dat een boek in de bibliotheek aanwezig moet zijn om hem daarvandaan te lenen, is een voorbeeld van een ‘alethic rule’. Het is namelijk onmogelijk om een boek dat niet aanwezig is toch mee te nemen.

2.2.5 Terminologische definities

De laatste ‘deliverable’ die van belang is bij een proces van requirements engineering, is die van de terminologische definities. De verzameling van al deze definities - de zogenaamde ‘glossary’ - zou je kunnen vergelijken met een woordenboek. Het is hierbij de bedoeling dat er van alle termen die voorkomen in alle domeinmodellen van een requirements analyse definities opgesteld worden. Een domeinmodel zelf geeft meestal alleen de relaties tussen concepten weer. Nergens staat wát een concept eigenlijk precies is. Dat is waar terminologische definities toe dienen.

Net als bij business rules zijn er talen om zulke definities precies op te schrijven. Ook kun je van veel termen een bestaande woordenboekdefinitie opzoeken. Toch werkt het misschien wel het beste om de definities zelf op te stellen. Het is een goed idee om hierbij de veelgebruikte structuur van een ‘definiens’ (hetgeen dat gedefinieerd wordt), een ‘genus’ (het supertype van wat gedefinieerd wordt) en één of meer ‘differentiae’ (de onderscheidende eigenschap van het gedefinieerde) te gebruiken. Een voorbeeld van een zo opgebouwde definitie is de volgende: “Een hond (definiens) is een dier (genus) dat kan blaffen (differentium) en met zijn staart kwispelt (differentium)”. Door alle termen in een requirements analyse zo op te stellen, voorkom je spraakverwarring en definities die door elkaar lopen. Zo zijn terminologische definities een nuttige toevoeging aan de voorgaande producten.

3. Het raamwerk

De tweede deelvraag betreft het definiëren van een eigen raamwerk, waarmee we de strategieën van de beschreven operationalisatie van requirements engineering kunnen specificeren. De basis van dit raamwerk ligt bij de doelen die de requirements engineer voor ogen heeft. Naar aanleiding van deze doelen kiest hij de te volgen strategieën. Het raamwerk moeten om kunnen gaan met verschillende smaken van strategieën. Zo moeten we een strategie open kunnen laten voor eigen interpretatie, of juist wel concreet invullen. Daarnaast moeten we in kunnen zoomen op strategieën en deterministische procedures erbij kunnen betrekken. Ook zullen strategieën soms in een bepaalde volgorde uitgevoerd moeten worden, en komt er dus een temporeel aspect bij kijken.

Dit zijn semantische eisen aan het raamwerk, maar er komen ook syntactische eisen aan te pas. Zo moeten we manieren vinden om het geheel aan strategieën op te schrijven en overzichtelijk te houden. We denken dat het uiteindelijke raamwerk aan al deze eisen voldoet.

Natuurlijk is het zo dat dit uiteindelijke raamwerk gedurende het onderzoek zelf flink geëvolueerd is. Het is namelijk schier onmogelijk om vooraf een perfect raamwerk te definiëren. Tijdens het creëren van de uiteindelijke specificatie zijn we er meerdere keren achter gekomen dat het raamwerk dingen miste. In zulke gevallen hebben we het raamwerk gewoon weer aangepast en geconcretiseerd.

Hierbij is het zo dat sommige kleine aanpassingen kunnen resulteren in veel grotere koerswijzigingen. In het oorspronkelijke raamwerk hadden we er bijvoorbeeld voor gekozen de strategieën weer te geven in een graph. Na meerdere keren ontdekt te hebben dat we daar niet alles uit konden drukken wat we wilden, hebben we besloten te gaan werken met de procestaal YAWL (zie sectie 3.3). Daarnaast is het ook zo dat we in sommige gevallen vooraf juist te veel in detail zijn getreden. We hadden namelijk concepten bedacht die we in de praktijk helemaal niet nodig bleken te hebben. Zo resulteerde het daadwerkelijke specificeren van de strategieën in een hoop toevoegingen en verwijderingen binnen het raamwerk.

We zullen nu allereerst de motivatie en achterliggende gedachten over de vorm van het raamwerk beschrijven. Na de motivatie definiëren we het uiteindelijke raamwerk zelf. Dit doen we in woorden en afbeeldingen, en we sluiten deze definitie af met een metamodel in de conceptuele modelleertaal ORM. Hierin komen alle eerder besproken concepten en relaties daartussen terug.

Vervolgens zullen we bespreken hoe dit raamwerk zich verhoudt tot de ‘state of the art’. Dit betreft de derde deelvraag; de literatuurstudie. Hierbij dient allereerst gezegd te worden dat dit afstudeeronderzoek nauw aansluit en voortbordurt bij lopend onderzoek aan onze eigen universiteit. Zoals in het artikel hierover (Van Bommel et al., 2007) ook gezegd wordt, was het hierin beschreven raamwerk vooral gebouwd op theoretische gronden. In dit artikel wordt tevens gezegd dat dit oorspronkelijke raamwerk nog getest en toegepast moet worden in de praktijk. En dat is nu juist waar ons afstudeeronderzoek toe dient.

Naast dit artikel van onze eigen universiteit hebben we nog een aantal artikelen gevonden, waarin de auteurs ook een soort raamwerken presenteren waarmee concrete processen zou kunnen worden gespecificeerd. Aan de hand van al deze artikelen uit het vakgebied van de ‘method engineering’ zullen we aangeven in hoeverre we (bijvoorbeeld qua terminologie) meegaan met hen, maar juist ook in hoeverre we bewust afwijken van wat zij beschrijven. We hebben er voor gekozen om deze vergelijking met andere, soortgelijke onderzoeken pas na de voltooiing van ons eigen raamwerk uit te voeren en dus zelf ‘vanaf nul’ te beginnen. Het is namelijk lang niet altijd slecht om het wiel even opnieuw uit te vinden. Zo voorkomen we dat we uitkomen op een bijeengeraapt raamwerk dat verre van consistent is. Voordat we deze literatuurstudie uitvoeren, zullen we dus eerst ons eigen uiteindelijke raamwerk presenteren.

3.1. Motivatie vorm van het raamwerk

We zullen nu een raamwerk op gaan stellen waarin we alles op kunnen nemen wat we over strategieën kwijt zouden willen. Zo zouden we bijvoorbeeld kunnen zeggen dat het in een bepaalde situatie slim zou kunnen zijn om eerst een use case op te stellen, alvorens we een scenario op gaan stellen. Hier gaat het dan vooral om de volgorde waarin je taken uit zou kunnen voeren.

We willen echter ook beschrijven hoe je een scenario op zou moeten stellen. Daar gaat het dan vooral om de manier waarop iets dient te gebeuren. Ook ligt het voor de hand om deze soorten van beschrijvingen (over de volgorde en over de manier waarop) niet helemaal los van elkaar te specificeren. Dit resulteert daarom in een raamwerk met meerdere niveaus, waarbij je op het hoogste niveau bijvoorbeeld enkel spreekt van ‘een use case opstellen’.

Op een lager niveau spreek je dan van stappen als ‘zorgen voor input van klant’ en ‘basic course of events opstellen’. Het is dan mogelijk om weer op de laatste stap in te zoomen, mocht dat noodzakelijk zijn. Dit proces kun je doorzetten tot je op een heel laag niveau terecht komt, waarbij de stappen niet meer zijn dan triviale instructies als ‘zoek onderwerp in feitzin’. De acties op dit niveau zijn vaak heel goed uit te voeren door werknemers. De problemen zitten juist in het vinden van acties op dit niveau en het bepalen van de volgorde van stappen die je moet nemen. Waar moet je bijvoorbeeld mee verder gaan als je net een use case gemaakt hebt? Ons raamwerk zal daar een onderbouwd antwoord op geven.

In de jonge geschiedenis van het vakgebied van de software ontwikkeling zijn er diverse pogingen gedaan om specificaties op te stellen volgens de zogenaamde ‘kookboek’-benadering. Bij een dergelijk raamwerk wordt de gebruiker ervan op een nogal deterministische wijze opgedragen wat hij of zij moet doen:

1. Verwarm de oven voor op 180 graden.
2. Snijd de geschildde aardappelen in plakjes.
3. Kook de aardappelplakjes in ruim kokend water met zout 3 minuten.
4. Was de spinazie grondig in ruim koud water, verwijder harde, dikke stelen en laat de spinazie goed uitlekken.

Op eenzelfde wijze werd in die methoden voor software ontwikkeling uiteengezet hoe je een use case moest opstellen. Deze stappenplannen (merk ook de nummering op in het voorbeeld) blijken in ons vakgebied niet te werken (Mirbel & Ralyté, 2006). Hier zijn meerdere redenen voor te noemen.

Allereerst is het zo dat een dergelijk stappenplan veel te deterministisch is. Het is niet per definitie verkeerd om sommige strategieën open te laten voor eigen invulling. Het heeft geen enkele zin om gebruikers van het raamwerk te gaan verplichten om feitzinnen te gaan ontleden, terwijl ze misschien zelf wel een veel betere methode hebben ontwikkeld om tot een ‘basic course of events’ te komen. Vaak doden deterministische procedures de creativiteit. Met ad-hoc beschrijvingen is in zulke gevallen dan ook niets mis. Het begrip ‘agility’ is hier van toepassing, oftewel de lenigheid van het proces. Een stappenplan kan hier niet goed mee overweg, net als met het geven van voorbeelden. Het is simpelweg te rigide.

Daarnaast is het zo dat in de softwareontwikkeling elk probleem, elke opdracht en dus ook elk proces van requirements engineering anders is. Je hebt met een andere beginsituatie te maken, er zijn misschien andere randvoorwaarden van toepassing, je hebt met andere mensen te maken die een andere functionaliteit voor het toekomstige systeem wensen en je hebt met een andere bedrijfscultuur te maken. Geen enkel proces van requirements engineering is dus hetzelfde. Requirements engineers zelf zijn natuurlijk ook gewoon mensen, en die verschillen van elkaar. Dus zelfs wanneer twee requirements engineers op hetzelfde moment een blauwdruk maken van de wensen van dezelfde klant, is de kans zeer groot dat er andere resultaten uitkomen.

Maar er zijn nog meer redenen waarom de methode van het ‘kookboek’ hier ongewenst is. Men richt zich hiermee vaak enkel op producten, en de procesmatige kant van de zaak is dan het ondergeschoven kindje. In dit onderzoek focussen we zelf op strategieën en daarmee dus ook op de processen. Maar daarnaast spelen de producten waar die processen tot leiden toch ook een heel belangrijke rol. Het raamwerk moet hier allemaal mee om kunnen gaan. Tenslotte zou het raamwerk ook om moeten kunnen gaan met plotselinge koerswijzigingen. Hiermee doelen we voornamelijk op een ander afstudeeronderzoek van Jorg Mutter dat ongeveer gelijktijdig met het onze is uitgevoerd, waarbij een zogenaamde modelleeragenda is geïmplementeerd. Om uit te leggen wat we hiermee bedoelen werkt een analogie met de TomTom GO verhelderend.

De TomTom GO is een navigatiesysteem voor in de auto, die na opgave van een bestemming via GPS en een ingebouwde wegenkaart bepaalt hoe je moet rijden om op die bestemming te geraken. Dit navigatiesysteem kan goed omgaan met plotselinge veranderingen. Stel dat er een boom op een weg valt waar je eigenlijk langs moet. Je moet daardoor een andere route nemen dan het systeem bedacht had. De TomTom GO rekent dan opnieuw uit hoe je vanuit die andere positie op een andere manier bij je bestemming kan komen. Op dezelfde manier werkt het genoemde systeem van de regelgebaseerde modelleeragenda. Ook deze genereert per situatie een eigen 'to-do list' en past deze automatisch aan veranderingen aan. Hoewel ons eigen afstudeeronderzoek hier los van staat, hebben we wel getracht om onze specificatie en ons raamwerk compatibel te houden met een dergelijk systeem doordat formele technieken als YAWL en ORM te gebruiken.

Al deze redenen waarom enkel het 'kookboek'-stappenplan ongeschikt blijkt te zijn om strategieën voor bijvoorbeeld requirements engineering te specificeren, houden we nu in het achterhoofd bij de opbouw van ons eigen raamwerk. Zoals gezegd sluit de definitie van dit raamwerk nauw aan bij lopend onderzoek van de afdeling IRIS aan onze eigen universiteit (Van Bommel et al., 2007). Ons raamwerk dat we in de volgende sectie presenteren is slechts gebaseerd op dezelfde basisprincipes. Het is juist een deel van deze afstudeeropdracht om dit raamwerk waar nodig aan te passen naar aanleiding van het gebruik ervan in de praktijk. En zo is ons eigen raamwerk dat we in de komende sectie presenteren een nuttige toevoeging op het lopende onderzoek.

3.2. Definitie van het raamwerk

Nu de achtergrond en motivatie van de keuze voor de vorm duidelijk is gemaakt, kunnen we gestalte gaan geven aan het raamwerk zelf dat we in deze masterscriptie hanteren. We maken in deze definitie onderscheid tussen strategieën en doelen, die erg met elkaar samenhangen. Het hoogste doel van een heel proces van requirements engineering is simpelweg het verkrijgen van de specificatie, van de requirements dus. De strategieën die je kunt volgen leiden uiteindelijk allemaal tot het voltooien van dit doel.

We hebben gekozen voor een raamwerk dat bestaat uit een verschillend aantal elementen. Zo geven we de context van een strategie weer middels een procestaal. De hieruit resulterende figuren zijn erg intuïtief, en niet alleen geschikt voor computerexperts of systeemontwikkelaars. Veel mensen kunnen zich gemakkelijk een voorstelling maken van wat er mee bedoeld wordt. Ook hebben ze een formele basis (petrinetten), waarmee we een dergelijke specificatie ook mogelijk houden. Daarnaast gebruiken we in het raamwerk tekstuele beschrijvingen en verhelderen we het geheel middels conceptuele modellen. We zullen alle elementen van het raamwerk in de komende subsecties de revue laten passeren. Merk op dat we bij de namen en definities van deze elementen die we invoeren ook gebruik maken van de Engelse termen en een schuin lettertype. Zo sluiten we beter aan bij het lopende onderzoek en benadrukken we de deels nieuwe terminologie.

3.2.1 Naam van een activiteit

We beginnen met de triviaal ogende opmerking dat we in ieder geval gebruik moeten maken van titels om het geheel van specificaties overzichtelijk te houden. Zoals eerder beschreven zullen we te maken krijgen met verschillende niveaus en lagen van strategieën. Het is onwenselijk en onmogelijk om dit in één grote beschrijving te gieten. Daarom zullen we de gehele specificatie op moeten splitsen. Elk los deel noemen we een activiteit of *activity*. Om naar een bepaalde activiteit te verwijzen dient gebruik te worden gemaakt van de exacte naam die we aan een activiteit geven.

Voorbeelden van zulke titels of namen – aflopend naar een steeds lager niveau – zijn 'Een requirements analyse maken', 'Een use case maken' en 'Een basis course of events maken'. Deze titels noemen we activiteitsnaam of *activity name*.

Grafisch gezien noteren we deze titels als volgt:

Een requirements analyse maken

Tabel 3: grafische weergave van een *activity name*

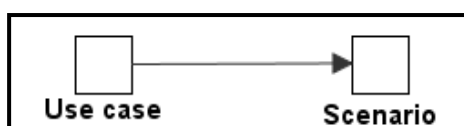
Het is belangrijk dat elk van deze namen uniek is, aangezien er naar verwezen moet worden. Het is namelijk goed voor te stellen dat er een tekstuele beschrijving van een strategie voor gaat komen die zo uitgebreid is dat we er een eigen *activity* voor aan moeten maken. Deze tekstuele beschrijving en de titel van de nieuwe activiteit moeten dan gelijk zijn, anders loopt het hele systeem van specificaties in de war. We gebruiken met opzet het woord ‘systeem’, aangezien we de specificaties uit het volgende hoofdstuk bij gaan houden via een online database. Via deze techniek kunnen we ook formeel afdwingen dat de naamaanduidingen uniek zijn. Een activiteit bestaat dus uit een activiteitsnaam, maar heeft daarnaast nog meer velden. In volgorde van verschijning zijn dit de strategie context, de strategie zelf en een gemodelleerde weergave van de concepten en relaties van een strategie. We zullen ze in de komende subsecties nader toelichten.

3.2.2 Context van de strategie

De kern van de zogenaamde *strategy context* bestaat uit minimaal twee toestanden of *states*, die aan elkaar gelinkt worden door middel van een transitie of *transition*. Toestanden zijn eigenlijk gewoon resultaten van stappen in een proces. Er zijn twee smaken van toestanden mogelijk. Enerzijds heb je de bronnen of *sources*, die eigenlijk neerkomen op de input van een bepaalde activiteit. Een bijzondere vorm hiervan is ruw materiaal of *raw material* (extra input zoals notities). Anderzijds heb je de producten of *products*, waarmee we het resultaat van een activiteit bedoelen. Bijzondere vormen hiervan zijn tussenproducten of *intermediate products* (extra output zoals testversies). De context van een strategie definiëren we dus simpeler gezegd als een overgang van een beginsituatie naar een doelsituatie. Een beginsituatie kan bijvoorbeeld op hoog niveau zijn ‘use case’, en de doelsituatie is dan om daar een ‘scenario’ uit te krijgen.

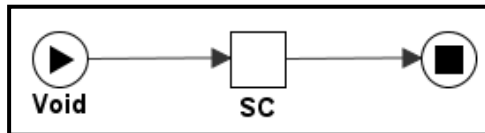
Zoals eerder gezegd gebruiken we de procestaal YAWL (Van der Aalst & Ter Hofstede, 2005) om de context van de strategie weer te geven. De afkorting YAWL staat voor ‘Yet Another Workflow Language’. Deze taal waarmee je workflows kunt maken, is ontwikkeld door de ‘Business Process Management’ onderzoeksgroep van ‘Queensland University of Technology’ in samenwerking met de technische universiteit van Eindhoven. De taal combineert inzichten uit onderzoek naar workflow patronen met het mechanisme van petrinetten. Het resultaat is een krachtige taal die in zijn fundamentele dusdanig simpel is dat het voor veel mensen erg gemakkelijk te begrijpen is.

Hoewel YAWL zeer veel (grafische) mogelijkheden biedt om dingen uit te drukken, gebruiken we in ons raamwerk slechts een subset van de notaties van deze procestaal. Daarbij komt dat we de taal voor iets anders gebruiken dan waar hij in essentie voor bedoeld is. YAWL modelleert welke taken in welke volgorde en in welke samenhang uitgevoerd dienen te worden. Wij zullen deze taken echter meer als toestanden interpreteren. De relaties kunnen we in het kader van dit afstudeeronderzoek titels geven als ‘heeft nodig’. De *states* geven we in YAWL weer als een vierkant, de transitie geven we weer als een pijl. Wanneer we uitgaan van de beschreven specificatie met een use case en een scenario, dan ziet dit er uit als in figuur 7.



Figuur 7: grafische weergave *strategy context*

In dit voorbeeld zien we aan de linkerkant van de transitie de bron en aan de rechterkant van de transitie het product, waar we de creatie van beschrijven. Er kleeft ook een temporeel aspect aan de bron en het product in dit model. Het creëren van de use case is eerder gebeurd dan het creëren van het scenario. Het kan echter ook voorkomen dat binnen een requirements engineering proces een product enkel met kennis en dus zonder expliciete, gestructureerde input wordt gecreëerd. Om dit aan te geven gebruiken we de programmeerterm 'Void'. Zie hiervoor figuur 8. Hierbij zie je ook direct het symbool dat je gebruikt om aan te geven dat de stroom van processen eindigt.



Figuur 8: grafische weergave *strategy context* met 'Void', stopconditie en afkorting

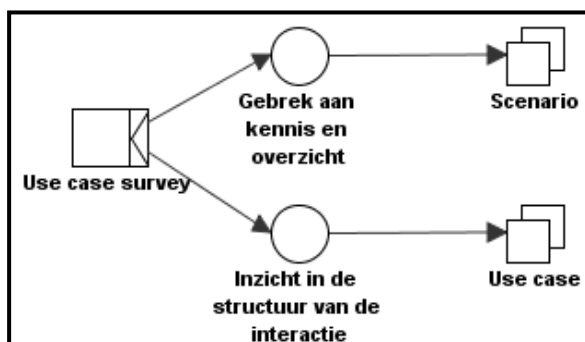
Merk op dat we hierin voor het product 'scenario' de afkorting 'SC' gebruiken. Dit gebruik van afkortingen heeft niet onze voorkeur, maar kan in bepaalde situaties met erg lange of veelvoorkomende namen van producten handiger zijn. Wanneer we afkortingen gebruiken voor de situaties, zorgen we ervoor dat deze helder gedefinieerd staan in een lijst met afkortingen. Voor de hele specificatie – die uit meer van deze graphstructuren bestaat – gebruiken we slechts één afkortingenlijst. Die ziet er bijvoorbeeld als volgt uit:

Afkorting	Betekenis
SC	Scenario
UCS	Use case survey

Tabel 4: lijst met afkortingen van voorbeeldspecificatie

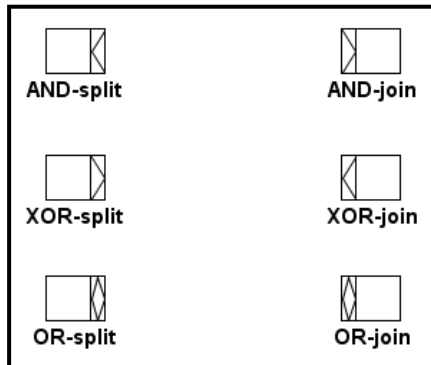
In hoofdstuk 8 van dit document is een uitgebreidere lijst met zulke afkortingen te vinden. Nogmaals, zelf prefereren we het gebruik van afkortingen niet, maar we kunnen ons voorstellen dat anderen dit wel doen. Daarom opperen we het bovenstaande gebruik ervan. Zelf streven we ernaar om dit in de specificatie in het volgende hoofdstuk achterwege te laten.

Verder heeft de taal YAWL nog enkele mogelijkheden die we via bijvoorbeeld een reguliere graph niet uit kunnen drukken. Dit betreft het weergeven van splitsingen of *splits* en van condities of *conditions* daarop. Wanneer je bijvoorbeeld een 'use case survey' gemaakt hebt, zijn er twee aan te raden opties om verder te gaan. Je kunt eerst een scenario gaan maken en vervolgens een use case, of juist andersom. Wanneer je als requirements engineer veel inzicht hebt in de interacties is de laatste optie aan te raden. Wanneer het je daaraan ontbreekt, is het aan te raden om eerst maar eens even een scenario op te stellen. Maar beide activiteiten dienen in ieder geval wel uitgevoerd te worden. Binnen YAWL geven we deze splitsing weer middels een iets andere notatie van de taak waar de splitsing vanuit gaat. De condities geven we weer als bolletjes met tekst. Zie figuur 9.



Figuur 9: grafische weergave *split* met *conditions* en *multiple tasks*

Merk op dat deze splitsing in het jargon van YAWL een 'AND-split' betreft, waarbij alle uitgaande taken stuk voor stuk uitgevoerd dienen te worden. Het is ook mogelijk om een 'OR-split' weer te geven. Hierbij maakt het niet uit of je één of meerdere of alle uitgaande taken uitvoert, zolang je er maar op zijn minst één uitvoert. Tenslotte heb je ook nog een 'XOR-split'. Hierbij wordt je verplicht één van de uitgaande knopen te kiezen, meer dan één van de opgesplitste taken kiezen is niet toegestaan. Daarnaast is het ook nog mogelijk om van elke van de drie varianten een 'JOIN' te definiëren. De grafische notaties hiervan zie je in figuur 10.



Figuur 10: grafische weergave soorten *splits* en *joins*

Voor het opstellen van deze modellen maken we gebruik van de officiële YAWL-editor (zie yawl-system.com). Hiermee biedt YAWL ons ook nog de mogelijkheid om meerdere taken of *multiple tasks* te definiëren. In figuur 9 was dit al te zien bij het scenario en de use case. Bij de grafische weergaven van deze taken zie je er eigenlijk meerdere achter elkaar staan. En dat klopt ook. Het is niet zo dat je na het voltooien van de use case survey slechts één use case gaat maken. Bijna altijd zullen er meerdere use cases gecreëerd moeten worden. Door een taak als een *multiple task* weer te geven in de *strategy context* weet de lezer ook meteen dat deze activiteit meerdere keren uitgevoerd dient te worden voor elke use case die gemaakt dient te worden. Hiermee komen we meteen terecht bij het derde deel van een activiteit, dat dus na de *activity name* en de *strategy context* komt. We hebben het over de strategie zelf.

3.2.3 Strategie

Het derde veld van een activiteit betreft de strategie zelf. Hierin wordt aangegeven hoe we nu daadwerkelijk vanuit een beginsituatie naar een doelsituatie kunnen gaan. In termen van het laatst gebruikte voorbeeld dus welke stappen we moeten ondernemen om een use case te creëren als we reeds een use case survey in elkaar hebben gezet.

Een strategie of *strategy* definiëren we als een verzameling van stappen of *steps* die ervoor zorgt dat de transitie uit de context van de strategie wordt uitgevoerd. Dit deel van de activiteit kan grafisch worden weergegeven door een opsomming van tekstuele regels. Zie hiervoor bijvoorbeeld tabel 5, waarin wat regels staan die van belang kunnen zijn bij het creëren van een use case survey.

- Actoren identificeren
- Een lijst van interacties opstellen
- Namen en beschrijvingen toevoegen

Tabel 5: grafische weergave van *steps* zonder *order*

Deze strategie bestaat dus uit drie stappen, die in dit geval geen volgorde hebben. Dit hoeft echter niet altijd zo te zien. Dit bespreken we in de volgende alinea's.

De volgorde van stappen

Net zoals bij de *strategy context* hebben we bij *steps* ook te maken met een temporeel aspect van volgorde. Bij de context van de strategie was deze volgorde vooral impliciet aanwezig door de gerichte pijlen die we voor de transities gebruikten. In de *strategy* geven we het verschil weer op een gedetailleerdere manier in de tekstuele beschrijving. Allereerst kunnen stappen géén volgorde of *no order* hebben zoals dat in tabel 5 al het geval was. In sommige gevallen maakt het namelijk helemaal niet uit in welke volgorde je stappen uitvoert, of wil je deze volgorde juist vrijlaten. Het is dan alleen belangrijk dát de stap uitgevoerd wordt. Deze opeenvolging van stappen geven we hier weer met streepjes als opsommingsteken.

Daarnaast kan het ook zo zijn dat we juist wel een volgorde af willen dwingen. Dit doen we op een niet zo schokkende wijze, zoals weergegeven is in tabel 6. Het is mogelijk om deze *order* op een meer wiskundige wijze vast te leggen. Wanneer we bijvoorbeeld de drie stappen voor het gemak even A, B en C noemen, ziet de *order* er dan als volgt uit: { <A,B>, <B,C> }. Hiermee wordt aangeduid dat stap B op stap A volgt en dat stap C op stap B volgt.

- | |
|--|
| <ol style="list-style-type: none"> 1. Een lijst van interacties opstellen 2. Actoren identificeren 3. Namen en beschrijvingen toevoegen |
|--|

Tabel 6: grafische weergave van *steps* met *order*

Er bestaan ook situaties waarbij je een bepaalde stap pas uit dient te voeren nadat twee andere stappen uitgevoerd zijn. Het wordt nog ingewikkelder als deze twee stappen geen volgorde hebben. Dit soort situaties geven we weer zoals in tabel 7. Hierbij maakt het niet uit welke stap 1 je als eerste uitvoert, maar wel dat stap 2 pas uitgevoerd wordt nadat de eerste twee stappen afgerond zijn.

- | |
|--|
| <ol style="list-style-type: none"> 1. Een lijst van interacties opstellen 1. Actoren identificeren 2. Namen en beschrijvingen toevoegen |
|--|

Tabel 7: grafische weergave van *steps* met dubbele *order*

Ook is het zo dat er zich een situatie kan voordoen waarin we binnen de strategie een splitsing willen weergeven. Hoewel deze constructie eigenlijk bedoeld is voor de *strategy context*, willen we dit in schaarse gevallen ook in de stappen aangeven. Wanneer we te maken hebben met een splitsing in meerdere, opeenvolgende taken, maken we gebruik van een *extended step* of uitgebreide stap. Hierbij geven we de opeenvolgende stappen als één stap weer en verbinden we de stappen middels het verbindingswoord 'EN'. Een voorbeeld van een dergelijke situatie - die in de uiteindelijke specificatie ook voorkomt - is te zien in tabel 8.

- | |
|--|
| <ol style="list-style-type: none"> 1. Een use case maken vanuit een use case survey EN Een scenario maken vanuit een use case 1. Een scenario maken vanuit een use case survey EN Een use case maken vanuit een scenario 2. Een domeinmodel maken |
|--|

Tabel 8: grafische weergave van *extended steps*

Vervolgens kan het ook zo zijn dat het van groot belang is dat een bepaald product direct na de voltooiing van een ander product gecreëerd dient te worden.

Dit is een duidelijk sterkere volgorde dan de eerder beschreven volgorde, omdat je in dit geval niet tussendoor ergens anders mee aan de gang kunt gaan. Het is in dat geval dus niet toegestaan om tussen deze twee stappen nog een stap zonder *order* te verrichten, terwijl dat bij een stap met een gewone *order* (zie tabel 6 en 7) wel toegestaan is. Deze onmiddellijke volgorde of *immediate order* geven we weer als in tabel 9.

1. Actoren identificeren
 - *2. Namen en beschrijvingen toevoegen

Tabel 9: grafische weergave van *steps* met *immediate order*

Merk hierbij op dat het toevoegen van een *immediate order* eigenlijk alleen zin heeft wanneer er ook stappen zonder *order* in de strategie aanwezig zijn. Anders gedraagt de stap zich vrijwel hetzelfde als een gewone *order*. Een korte samenvatting van de drie soorten volgorden is te vinden in tabel 10.

Term	Beschrijving	Weergave
<i>No order</i>	De volgorde waarin stappen uitgevoerd worden maakt niet uit.	- Stap A - Stap B
<i>Order</i>	Een stap met een hoger lijstnummer dient na een stap met een lager lijstnummer uitgevoerd te worden, eventueel met tussenkomst van stappen zonder <i>order</i> .	1. Stap A 2. Stap B
<i>Immediate order</i>	Een stap met een hoger lijstnummer dient onmiddellijk na een stap met een lager lijstnummer uitgevoerd te worden, zonder tussenkomst van stappen zonder <i>order</i> .	1. Stap A *2. Stap B

Tabel 10: samenvatting drie soorten van *order*

Merk op dat deze volgorden ook door elkaar kunnen lopen binnen één *strategy context*. Zie hiervoor tabel 11. Hierbij is het een harde eis dat stap B en stap C direct na elkaar volgen. Stap A mag helemaal in het begin van deze volgorde en helemaal aan het eind. Stap A mag echter ook tussen stap C en stap D geschieden, zolang stap D maar wel ná stap C volgt. Hieraan is al te zien dat via deze streepjes, nummers en sterretjes best complexe structuren in elkaar te zetten zijn.

Stappen	Legale volgordes
- Stap A	<A, B, C, D>
1. Stap B	<B, C, D, A>
*2. Stap C	<B, C, A, D>
3. Stap D	

Tabel 11: grafische weergave van *steps* met verschillende soorten *order*

Het is belangrijk om op te merken dat we in onze eigen specificatie (zie hoofdstuk 4) de *immediate order* niet nodig hebben gehad. Ook bij stappen die helemaal geen volgorde leken te hebben, hebben we er in de praktijk toch voor gekozen om deze met een normale volgorde weer te geven.

Het niet gebruiken van deze twee soorten volgorden heeft te maken met de losse manier van hanteren waar wij in dit onderzoek voor gekozen hebben. Wanneer we te maken zouden hebben met striktere toepassingen - zoals het systeem van de modelleeragenda (beschreven in sectie 3.1) - is het goed voor te stellen dat we deze expliciete volgorderegulering wél dienden te gebruiken. Daarom laten we deze soorten volgorde hier staan en laten we het gebruik ervan over aan anderen.

De soorten stappen

Er bestaan drie soorten stappen. Allereerst hebben we de *ad-hoc step*. In de motivatie van dit raamwerk (sectie 3.1) hebben we beargumenteerd dat het mogelijk moet zijn om niet verder in te zoomen op een stap of strategie, zodat we niet in een deterministische situatie terecht komen. De reden hiervoor kan zijn om zo de creativiteit van de requirements engineer te bevorderen. Ook kan het zo zijn dat het gewoonweg niet nodig is om de stap op een dieper niveau uit te werken, omdat het huidige niveau al duidelijk genoeg gespecificeerd is. Alle stappen die tot nu toe in de voorbeelden de revue zijn gepasseerd betreffen ad-hoc stappen. Qua opmaak is hier dus niets bijzonders aan, de stap is niet dikgedrukt en niet onderstreept.

De tweede soort stap is de *guided step*. Hierbij is het mogelijk om tips of *notes* bij bepaalde stappen aan te geven. Hierin kunnen we bijvoorbeeld middels vrije tekst aangeven wat er bij een stap handig is, waar de requirements engineer aan moet denken en wat belangrijke kwaliteitscriteria bij een dergelijke stap zijn. Een begeleidende stap geven we weer door de stap te onderstrepen en eronder een opsomming van tips te geven. Hiervoor gebruiken we deze keer bolletjes (of *bullets*) als opsommingtekens, zie hiervoor tabel 12.

1. Een lijst van interacties opstellen

- Definieer zoveel mogelijk interacties
- Win informatie in bij meerdere stakeholders
- Kies sterke werkwoorden om de interacties te beschrijven

Tabel 12: grafische weergave van een *guided step*

Tenslotte kan de stap samengesteld of *composed step* zijn. Dit betekent dat de stap verder uitgediept dient te worden. Voor dit soort stappen definiëren we wederom een andere notatie. We geven deze stappen namelijk weer in een dikgedrukt (meestal 'bold' genoemd) lettertype. Hieruit is af te leiden dat er nog meer te vinden is over deze stap dan alleen de tekstuele regel in deze strategie zelf. Een voorbeeld van een dergelijke samengestelde stap is te vinden in tabel 13.

- Een lijst van interacties opstellen
- Namen en beschrijvingen toevoegen
- **Een use case diagram opstellen**

Tabel 13: grafische weergave van een *composed step*

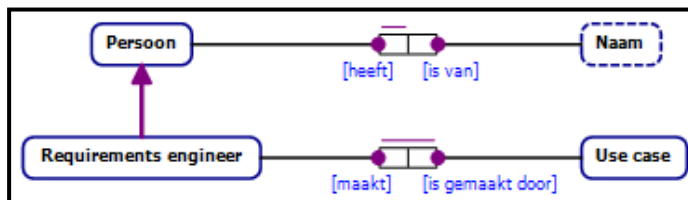
Deze samengestelde stap leidt dus tot een nieuwe activiteit. Hierin zoomen we verder in op deze specifieke activiteit. Hiervoor moeten we echter wel duidelijk maken waar deze ingezoomde activiteit dan te vinden is. En dit is nu precies de voorname reden van het bestaan voor de activiteitsnaam die we eerder beschreven in sectie 3.2.1. In de specificatie moet nu een activiteit te vinden zijn die ook de exacte titel 'Een use case diagram opstellen' heeft. Het betreft dus een vorm van recursie.

Merk op dat het aan te raden is om in de uiteindelijke specificatie al deze lagen van strategieën wel op volgorde te plaatsen. Daarmee beginnen we dan op het hoogste niveau bij een activiteit als 'Een requirements analyse maken'. Verder gaan we depth-first de hele specificatie af. We beginnen dus met de eerst voorkomende samengestelde stap (als die er is) uit deze activiteit, bijvoorbeeld 'Een problem statement maken'. Vervolgens gaan we weer verder met eventuele samengestelde stappen uit deze activiteit, voordat we naar de tweede stap van het hoogste niveau gaan.

We geven toe dat een dergelijke totale specificatie met verschillende lagen op papier niet ideaal is. Daarom presenteren we in sectie 4.2 een interactief, online systeem waarbij gebruikers gewoon op samengestelde stappen kunnen klikken (het zijn dan *hyperlinks*) en zo naar een pagina met de gevraagde ingezoomde activiteit kunnen gaan.

3.2.4 Conceptuele beschrijving

In de voorgaande subsecties hebben we besproken hoe de *name*, *strategy context* en *strategy* van een activiteit eruit zien. In deze subsectie bespreken we het laatste deel van de activiteit, waarin de concepten, relaties en beperkende regels daarover die in de activiteit aan bod zijn gekomen nader toegelicht worden. Daarnaast zouden we via deze conceptuele beschrijving software op onze specificatie los kunnen laten, waarmee automatisch gecontroleerd kan worden of er wel aan alle eisen of *constraints* voldaan is. Dit alles kan gedaan worden middels de eerder in sectie 2.2.3 over domeinmodellen genoemde modelleertalen. In de specificatie zullen wij wederom gebruik maken van de taal ORM en de modelleertool NORMA. Een dergelijke conceptuele beschrijving of *conceptual description* kan er dan uitzien zoals in figuur 11:

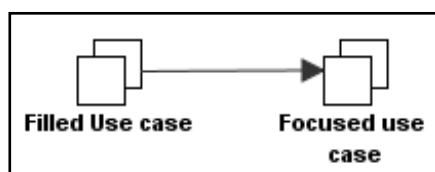


Figuur 11: grafische weergave *conceptual description*

3.2.5 Iteraties

Naast de hierboven uiteengezette definitie van ons raamwerk, willen we nog enkele aanvullende opmerkingen kwijt. Hiermee komen we in deze subsectie terecht bij het aspect van de iteraties. Het is een algemeen bekende veronderstelling in de softwareontwikkeling dat de zogenoemde 'Waterfall'-methode niet werkt (Kruchten, 2000). Dit is een methode van ontwikkelen waarbij er niet meer naar voorgaande producten en documenten wordt gekeken als het af is. Dit is echter een onhoudbare situatie. Tijdens een project verandert er zeer veel en doe je vaak nieuwe inzichten op. Hierdoor dienen voorgaande en afgeronde producten en documenten meerdere keren bijgewerkt te worden. Dit is ook essentieel voor de kwaliteit van het uiteindelijke product. Het is echter zo dat je elke iteratie van een of meer producten vanuit elke situatie in zou kunnen gaan. Dit betekent grafisch gezien dat je bijna vanuit bijna elke situatie een pijl richting elke andere situatie zou moeten tekenen.

Het is duidelijk dat dit te ver gaat en het geheel een stuk onleesbaarder maakt. Daarom nemen we dit aspect van iteraties niet mee in ons raamwerk. Een ander aspect van iteraties nemen we wél mee in ons raamwerk, maar hebben we zelf in de specificatie niet gebruikt. In sectie 2.2.1 hebben we reeds gesproken over een drietal stadia die vaak gebruikt worden om de status en levensloop van producten aan te geven. De eerste beginnende fase noemen we 'Facade', de tweede convergerende fase noemen we 'Filled' en de laatste divergerende fase noemen we 'Focused'. Dit principe is erg logisch en wordt ook vaak gebruikt om iteraties mee uit te drukken. Zo kunnen we een *strategy context* krijgen zoals hieronder in figuur 12 te zien is.

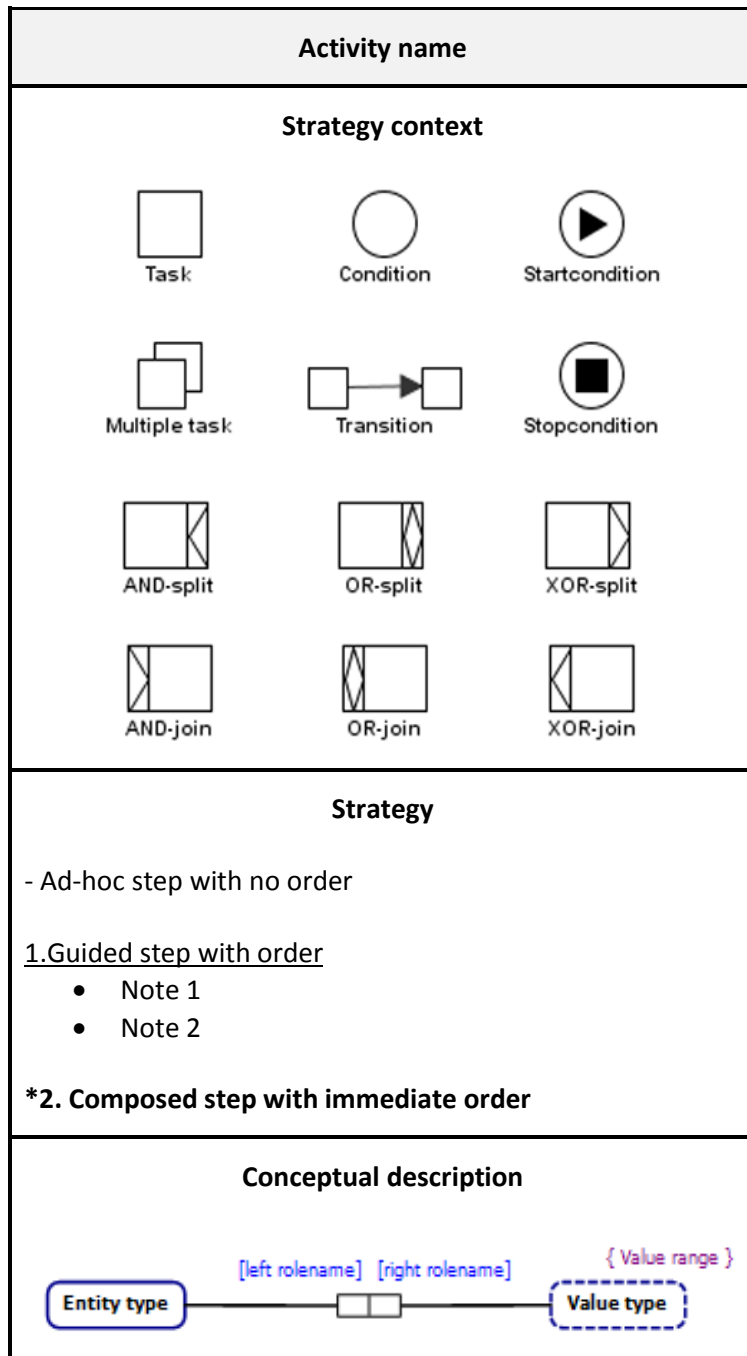


Figuur 12: grafische weergave expliciete iteraties binnen *strategy context*

Het bleek echter ook dat we dit mechanisme in onze eigen specificatie liever niet gebruikten. Dat is vooral te wijten aan het feit dat we een bepaalde manier van beschrijven in gedachte hebben gehouden. We vonden het toch te ver gaan om ook onderscheid te maken tussen verschillende stadia van producten. Maar ook hier geldt weer dat we ons goed kunnen indenken dat het in andere gevallen wél iets kan toevoegen. Twijfel dan vooral niet om het te gebruiken.

3.2.6 Een volledig opgebouwde activiteit

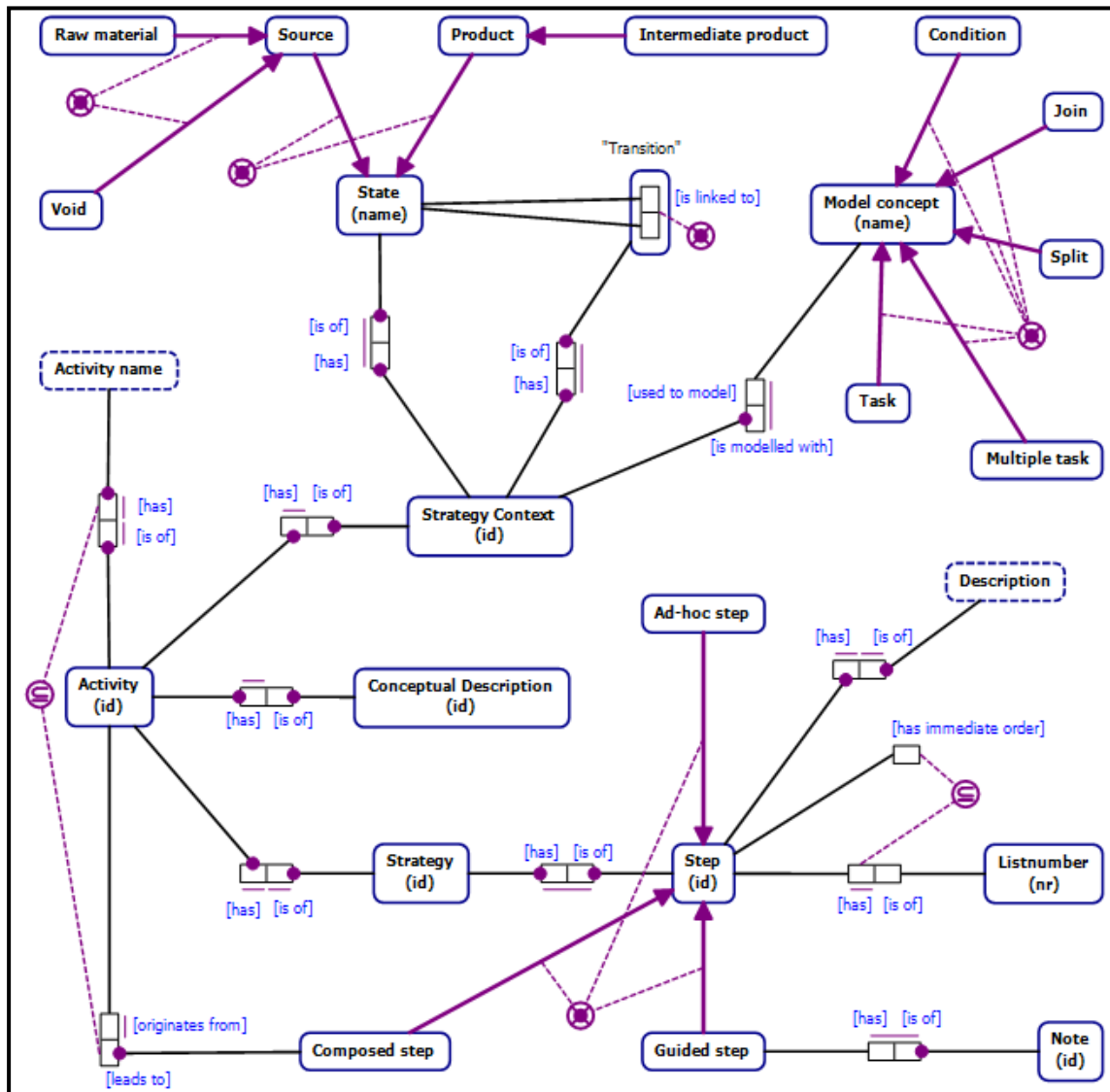
In voorgaande secties hebben we de definitie van ons raamwerk deeltje voor deeltje opgebouwd. In deze sectie presenteren we ten behoeve van het overzicht nogmaals het raamwerk, maar dan in één grote schematische weergave. Zie hiervoor de volledige *activity* in tabel 14.



Tabel 14: grafische weergave volledig opgebouwde activity

3.2.7 Meta-model

In deze laatste subsectie van de definitie completeren we ons raamwerk door een metamodel te presenteren van de concepten en relaties die in het raamwerk voorkomen. Zie hiervoor figuur 13. Het doel van dit metamodel is om de gebruiker het raamwerk zo goed mogelijk te laten begrijpen. Daarom is de database die we gebruiken voor het interactieve systeem niet helemaal opgebouwd volgens dezelfde concepten en relaties. Zie hiervoor wederom sectie 4.2.



Figuur 13: meta-model raamwerk

3.3. Plaatsing van het raamwerk via literatuurstudie

De derde deelvraag van dit afstudeeronderzoek zullen we beantwoorden via een literatuurstudie. We zijn hierbij voornamelijk op zoek gegaan binnen het vakgebied van de method engineering. Dit deel van de scriptie was aanvankelijk open-ended. We wisten niet in hoeverre er literatuur beschikbaar was over een onderzoek dat lijkt op wat wij hebben gedaan. Gelukkig hebben we wel enkele artikelen kunnen vinden. Daarmee kunnen we ons raamwerk plaatsen ten opzichte van 'the state of the art' en andere raamwerken, en hebben we lering kunnen trekken uit de daarin door andere onderzoekers gebruikte concepten en relaties.

We hebben er daarnaast ook bewust voor gekozen om deze literatuurstudie pas na de voltooiing van ons eigen raamwerk uit te voeren. Hierdoor zijn we wellicht tot op zekere hoogte bezig geweest het wiel opnieuw uit te vinden, maar dat hoeft niet altijd slecht te zijn. Op deze manier konden we een raamwerk opbouwen dat precies aan onze eigen eisen voldeed. Het resultaat is een consistent en compleet geheel.

In de komende secties laten we een aantal soortgelijke raamwerken de revue passeren. We hebben getracht een zo gevarieerd mogelijke verzameling van artikelen te selecteren. De titel van elke sectie is vernoemd naar de auteurs van de desbetreffende raamwerken. We zullen ze kort inleiden en ons voornamelijk richten op de overeenkomsten en verschillen ten opzichte van ons eigen onderzoek.

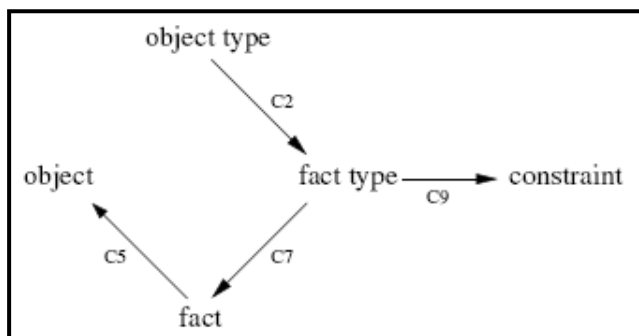
3.3.1 Van Bommel et al.

Het eerste artikel (Van Bommel et al., 2007) waarmee we ons raamwerk gaan vergelijken, is het artikel van onze eigen onderzoeksafdeling dat de aanleiding vormde voor dit afstudeeronderzoek. In dit artikel wordt de eerste aanzet gegeven tot het beschrijven van een raamwerk waar de afdeling IRIS van de Radboud Universiteit op uit is gekomen. Dit raamwerk was echter vooral gebouwd op theoretische gronden. Daarom is in de sectie 'conclusion and further work' het volgende te lezen:

"Finally, we are already engaged in four different method modelling exercises based on the framework presented in this paper. These activities will lead to validation, improvement, and refinement of the framework. One exercise concerns the method underlying a university Requirements Engineering course."

Hiermee wordt dus ons afstudeeronderzoek bedoeld en het raamwerk van toen is inderdaad een stuk verfijnd. De motivatie van het oorspronkelijke raamwerk en de doelen die we er mee willen bereiken zijn bijna hetzelfde gebleven. Echter, tijdens het werken met het raamwerk zijn we tegen diverse problemen aangelopen. Onder andere omdat we iets wilden uitdrukken dat niet via het oorspronkelijke raamwerk uit te drukken was. Voornamelijk in het begin van het opstellen van de specificatie was dit het geval. Naar het einde toe werkte het raamwerk in de praktijk door de aanpassingen steeds beter.

Het grootste verschil tussen het raamwerk van Van Bommel et al. en ons eigen raamwerk is te vinden in de *strategy context*. In het oorspronkelijke raamwerk werd gebruik gemaakt van een structuur met graphen om het verloop van processen weer te geven, zoals in figuur 14. De context van de strategie werd gedefinieerd als een combinatie van minimaal twee toestanden of *states* die verbonden waren middels een transitie. Hoewel in het artikel oorspronkelijk enkel over gerichte pijlen wordt gepraat, was het idee om ook ongerichte pijlen en dikgedrukte pijlen (om een *immediate order* aan te geven) te gaan gebruiken. Tijdens het in elkaar zetten van de specificatie bleek echter al snel dat we deze verschillende vormen van pijlen niet in de context nodig hadden, maar in de strategie zelf.



Figuur 14: graphstructuur raamwerk Van Bommel et al.

Daarnaast kwamen we erachter dat er een aantal mechanismen in deze graphstructuur ontbraken. Zo konden we niet uitdrukken op welke gronden je bij een opsplitsing voor een bepaalde pijl moet kiezen, zoals bij het kiezen voor het maken van een scenario óf van een use case na het voltooiën van een use case survey. Ook was er geen handig mechanisme om een zogenaamde ‘for each’ aan te geven, zoals we bij de strategie voor een use case aan willen geven dat deze voor elke use case opnieuw uitgevoerd dient te worden. We hadden toen het vermoeden dat we hier via de procesmodelleertaal YAWL verder mee zouden komen. In YAWL waren deze twee mechanismen inderdaad wél ingebouwd. Je kunt condities aangeven in bolletjes en een ‘multiple task’ weergeven. Daarnaast waren er nog een aantal extra mogelijkheden, zoals het weergeven van *joins* en *splits* in de smaken *AND*, *XOR* en *OR*. Daarom hebben we de graphen toen door YAWL-modellen vervangen.

De basis van de strategie is hetzelfde gebleven. Alleen hebben Van Bommel et al. een wat andere naamgeving dan wij. Zij hadden nog geen naam voor het geheel, wij noemen het een activiteit. Wat wij stappen noemen, noemen zij strategieën. Wij noemen juist het geheel aan stappen een strategie. Zij maken een bijna identiek onderscheid tussen de drie soorten stappen, alleen gebruiken zij de namen ‘*layered strategy*’ en ‘*procedural strategy*’, terwijl wij respectievelijk voor de namen ‘*composed step*’ en ‘*guided step*’ hebben gekozen.

Naast deze naamgeving is er een groter verschil tussen beide raamwerken. Zij geven aan dat bij hun procedurele strategie een concretere beschrijving te vinden is in de vorm van (gestructureerde) tekst, pseudo-code of een conceptueel model. Wij hebben besloten om altijd te kiezen voor natuurlijke taal. Gebruikers kunnen vaak niet zoveel met technische aanduidingen als pseudo-code. Ook hebben we ervoor gekozen om bij elke activiteit verplicht een conceptuele beschrijving toe te voegen. De relaties tussen de objecten waar in de strategie over gesproken wordt verhelderen de activiteit enorm. Daarom is het een belangrijke verbetering aan het oorspronkelijke raamwerk.

Tenslotte zijn er nog een aantal details veranderd. Zo hebben we de volgorde van stappen in enkele gevallen wat geavanceerder gemaakt, bijvoorbeeld door een soort *joins* en *splits* (zie tabel 7) en *extended steps* (zie tabel 8) in de strategie toe te voegen. Merk overigens op dat in het artikel van Van Bommel et al. veel aandacht is besteed aan het regelgebaseerde karakter van hun raamwerk. Hoewel wij er in deze masterscriptie nauwelijks aandacht aan besteden, zou je de strategieën in ons raamwerk wel degelijk middels dergelijke regels weer kunnen geven. Om het formaliseren van de modellen in YAWL en in ORM hoeven we ons echter niet druk te maken, zij bieden zelf al een formele ondergrond.

Het is nu duidelijk hoe en waarom ons raamwerk geëvolueerd is uit het raamwerk van Van Bommel et al. In de overige subsecties van dit hoofdstuk zullen we ons raamwerk plaatsen ten opzichte van de ‘state of the art’ die onafhankelijk van ons onderzoeksinstituut ontstaan is.

3.3.2 Mirbel en Ralyté

In het artikel “Situational method engineering: combining assembly-based and roadmap-driven approaches” (Mirbel & Ralyté, 2006) beschrijven de auteurs hun raamwerk voor ‘*situational method engineering*’. Het eerste verschil tussen hun raamwerk en het onze is de grootte van de beschrijving ervan. Mirbel en Ralyté hebben ervoor gekozen om een zeer uitgebreid raamwerk op te stellen, terwijl wij het wat bescheidener aan hebben gepakt.

Wij hebben het belangrijk gevonden dat de uit het raamwerk volgende specificatie goed klopt en helder beschreven kan worden. Aangezien we in de praktijk gezien hebben dat ons raamwerk aan deze eisen voldoet en prima functioneert, kunnen we de vraag stellen of het model van Mirbel en Ralyté niet wat té uitgebreid is.

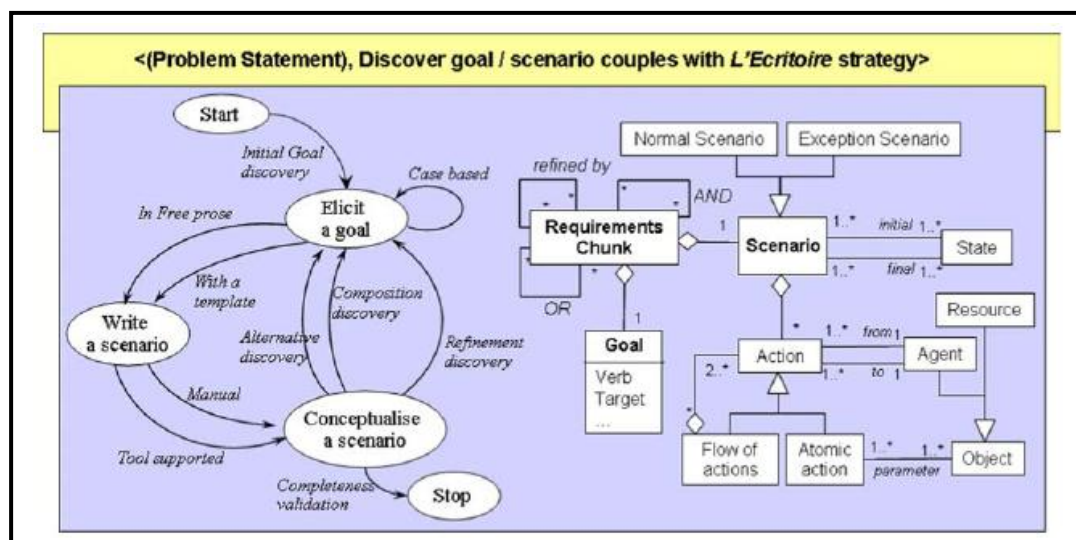
De elementen die de kern van hun raamwerk vormen, zijn de zogenaamde *'method chunks'*. Dit element representeert een herbruikbaar stuk bouw materiaal voor een methode. Het bevat zowel productfragmenten als procesfragmenten. Het productdeel van een *method chunk* definieert concepten, relaties daartussen en bijbehorende constraints. Het procesdeel beschrijft juist hoe je het betreffende product dient te maken. De oplettende lezer zal de treffende gelijkenis met ons eigen raamwerk al opgemerkt hebben.

Een *method chunk* komt in de buurt van wat wij een *activity* noemen, het procesdeel daarvan lijkt op onze *strategy context* en het productdeel daarvan staat gelijk aan onze conceptuele beschrijving. Een *chunk* is letterlijk een stukje van de methode, en dat zijn de activiteiten bij ons ook. De aanduiding of titel van deze *method chunks* is wel wat complexer, zij gebruiken zogenaamde *descriptors* (die de naam en de ID beschrijven) en *interfaces* (die de intentie en het doel beschrijven), terwijl wij als *activity name* eigenlijk alleen de intentie (zoals 'een use case maken') gebruiken.

De gelijkenis tussen beide raamwerken blijft nog even aanwezig, wanneer Mirbel en Ralyté hun definitie van *guidelines* introduceren. Hierover schrijven ze het volgende:

"A guideline is defined as 'a statement or other indication of policy or procedure by which to determine a course of action.' ... Guidelines in different methods have different contents, formality, granularity, etc. In order to capture this variety, three types of guidelines: simple, tactical and strategic are identified."

Ook dit komt in hoofdlijnen weer overeen met de drie soorten stappen in de strategie die wij beschrijven. Toch blijken Mirbel en Ralyté het hier net wat anders in te vullen. Zij gebruiken namelijk niet drie soorten stappen, maar drie soorten strategieën. Een *simple guideline* is een strategie waarin enkel gebruik wordt gemaakt van stappen in de vorm van (informele) tekst. Dat doen wij in elke strategie. Een *tactical guideline* gebruikt daarentegen een boomstructuur om diverse *sub-guidelines* (onze samengestelde stappen) aan op te hangen die steeds verder ontleed worden totdat je simpele instructies overhoudt. In grote lijnen zou je dit kunnen vergelijken met het idee van onze *guided steps*, hoewel wij deze gewoon in tekst weergeven en niet in boomstructuren. Een *strategic guideline* tenslotte gebruikt een graphstructuur om allerlei *sub-guidelines* (wederom onze samengestelde stappen) aan te duiden. Een voorbeeld van zo'n *strategic guideline* is te vinden in figuur 15. Deze modellen lijken weer het meest op onze YAWL-modellen, hoewel de uitdrukingskracht van de graphen een stuk minder is.



Figuur 15: *strategic guideline* raamwerk Mirbel en Ralyté

Het zal inmiddels duidelijk zijn dat het raamwerk van Mirbel en Ralyté een stuk complexer is dan ons eigen raamwerk. Enerzijds heeft de manier waarop zij het invullen wel enkele voordelen die wij niet hebben. Aangezien wij in onze strategie alleen maar gebruik maken van tekstuele stappen, hebben we wat constructies moeten bedenken om de volgorde van deze stappen goed weer te geven. Zij gebruiken hier verschillende vormen van modellen voor, die daar meer voor bedoeld zijn. Wij maken gebruik van dit soort modellen in de *strategy context* en in veel gevallen zou dat best in tekst weergegeven kunnen worden. In veel activiteiten bestaat ons model in YAWL enkel uit twee taken met één pijl ertussen. Dat zouden we dan beter kunnen weergeven met de *descriptors* en *interfaces* van hun raamwerk.

Toch is het zo dat wij in de strategie voldoende vrijheid wilden hebben om de stappen zo goed mogelijk te beschrijven. Als modelleur wil je hierbij niet bezig zijn met ingewikkelde boom- en graphstructuren. Dat maakt het tevens veel te ingewikkeld voor gebruikers die niet uit de ICT-hoek komen. Daarom denken we dat ons raamwerk veel intuïtiever is dan het raamwerk van Mirbel en Ralyté, hoewel de uitdrukingskracht van hun raamwerk veel groter is. En dan hebben we het nog niet eens gehad over de constructies die ze beschrijven om *method chunks* in op te slaan en te hergebruiken. Dit is vooral te wijten aan het feit dat Mirbel en Ralyté waarschijnlijk heel andere doelen voor ogen hadden dan wij. Wij wilden slechts een overzichtelijk en helder raamwerk in elkaar zetten waarmee we methodes als het proces van Requirements Engineering zo begrijpelijk mogelijk konden beschrijven. En zoals in hoofdstuk 4 te zien is, zijn we daar in geslaagd.

3.3.3 Van de Weerd et al.

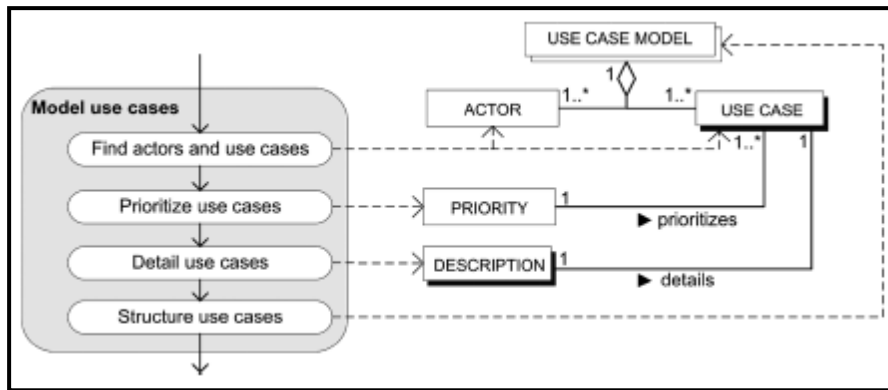
Aan de universiteit van Utrecht wordt ook veel onderzoek verricht rondom het vakgebied van de 'situational method engineering'. In het artikel "A Situational Implementation Method for Web-based Content Management System-applications: Method Engineering and Validation in Practice" presenteren Inge van de Weerd, Sjaak Brinkkemper, Juriaan Souer en Johan Versendaal een raamwerk dat wederom veel overeenkomsten met het onze vertoont (Van de Weerd et al, 2006). Dat betekent ook dat dit raamwerk veel lijkt op het raamwerk van Mirbel en Ralyté. Waarschijnlijk zijn er gewoonweg een aantal elementen waar je niet omheen kan, zoals een conceptuele beschrijving, een procesdiagram en het volgen van een aantal stappen. Maar dat kan ook te maken hebben met wederzijdse beïnvloeding, aangezien iedereen waarschijnlijk elkaars werk wel leest.

Van de Weerd et al. geven allereerst de volgende definitie van het vakgebied 'method engineering':

"... the engineering discipline to design, construct and adapt methods, techniques and tools for the development of information systems."

Vervolgens beschrijven ze hoe hun raamwerk opgebouwd is. De basis van het raamwerk is een techniek om metamodellen in elkaar te zetten die speciaal ontworpen is voor de 'method engineering'. Bij deze techniek wordt een zogenaamd proces-data-diagram opgesteld. Dit komt overeen met de notie van *method chunk* bij Mirbel en Ralyté en ook met onze eigen activiteiten, waar tevens een procesdeel en een productdeel in verenigd is. Van de Weerd et al. geven aan dat dit het mogelijk maakt om de relaties tussen processen en data in te zien en een methode vanuit beide perspectieven te bekijken. We zullen dit proces-data-diagram nu nader bekijken.

Het diagram bestaat uit twee geïntegreerde metamodellen. Aan de linkerkant vinden we een procesmodel, dat deze keer gebaseerd is op een 'UML activity diagram'. Aan de rechterkant van het proces-data-diagram vinden we een datamodel dat gebaseerd is op een 'UML class diagram'. Via stippellijnen wordt aangegeven welke dataconcepten gebruikt worden bij de diverse activiteiten in het procesmodel. Zo'n proces-data-diagram zie je in figuur 16.



Figuur 16: proces-data-diagram raamwerk Van de Weerd et al.

Van de Weerd et al. kiezen dus voor een raamwerk dat in zijn geheel uit modellen bestaat. Het datamodel komt vrijwel exact overeen met onze conceptuele beschrijving, afgezien van het feit dat wij ORM gebruiken en zij UML. Het procesmodel komt overeen met onze strategie, afgezien van het feit dat Van de Weerd et al. ze niet in tekst weergeven maar wederom gebruik maken van een model uit de UML-familie. Tevens geven ze de samenhang tussen beide modellen aan. Dit hebben wij in ons eigen raamwerk niet gedaan. De vraag is of het wat toevoegt, aangezien de relaties vaak wel heel erg voor de hand liggen.

In hun raamwerk hebben ze daarnaast wat kleine dingen toegevoegd aan de UML-modellen. Allereerst maken ze in het procesmodel ook gebruik van zogenaamde '*unordered activities*', om aan te geven dat er geen volgorde tussen de stappen zit. Dit komt exact overeen met de stappen zonder volgorde uit ons eigen raamwerk. Ook wij erkennen dat deze constructie handig kan zijn, hoewel het in onze specificatie juist niet nodig bleek te zijn.

Een andere aanpassing die Van de Weerd et al. aan het datamodel gedaan hebben, is het erkennen van meerdere soorten concepten. Ze maken allereerst gebruik van een *simple concept*, wat geen subconcepten bevat. Daarnaast definiëren ze een *open concept*, dat wél samengesteld is uit een open verzameling van één of meer (sub)concepten. Tenslotte noemen ze ook nog een *closed concept*, dat een niet uitbreidbare verzameling betreft. Middels deze drie soorten concepten wordt gelaagdheid in hun raamwerk toegevoegd. Waar zij deze gelaagdheid echter via de concepten in het datamodel laten lopen, hebben wij ervoor gekozen om juist de (samengestelde) stappen naar andere activiteiten te laten verwijzen. Vaak zijn deze verschillende keuzes vooral een kwestie van smaak en kleven er voor- en nadelen aan.

3.3.4 Overige raamwerken

We denken dat we met de voorgaande gedetailleerde bespreking genoeg overeenkomsten en verschillen tussen een uiteenlopende verzameling van drie raamwerken uit de method engineering en ons eigen raamwerk gezien hebben. We hebben hierbij tevens gezien dat veel raamwerken globaal gezien toch sterk op elkaar lijken, aangezien je gewoon niet zonder bepaalde elementen in je raamwerk heen kan. Dat maakt een vergaande analyse van nog meer raamwerken steeds minder interessant.

Het is goed om te zien dat ons raamwerk niet onder doet voor de 'state of the art' uit de method engineering. Het belangrijkste voor ons was echter een raamwerk te maken waar we activiteiten zoals het maken van een requirements analyse goed in kwijt konden. Het resultaat hiervan is volgens ons bevredigend. De totale specificatie van requirements engineering volgens ons raamwerk zoals het er op onze universiteit aan toe gaat is te vinden in het volgende hoofdstuk.

4. De specificatie

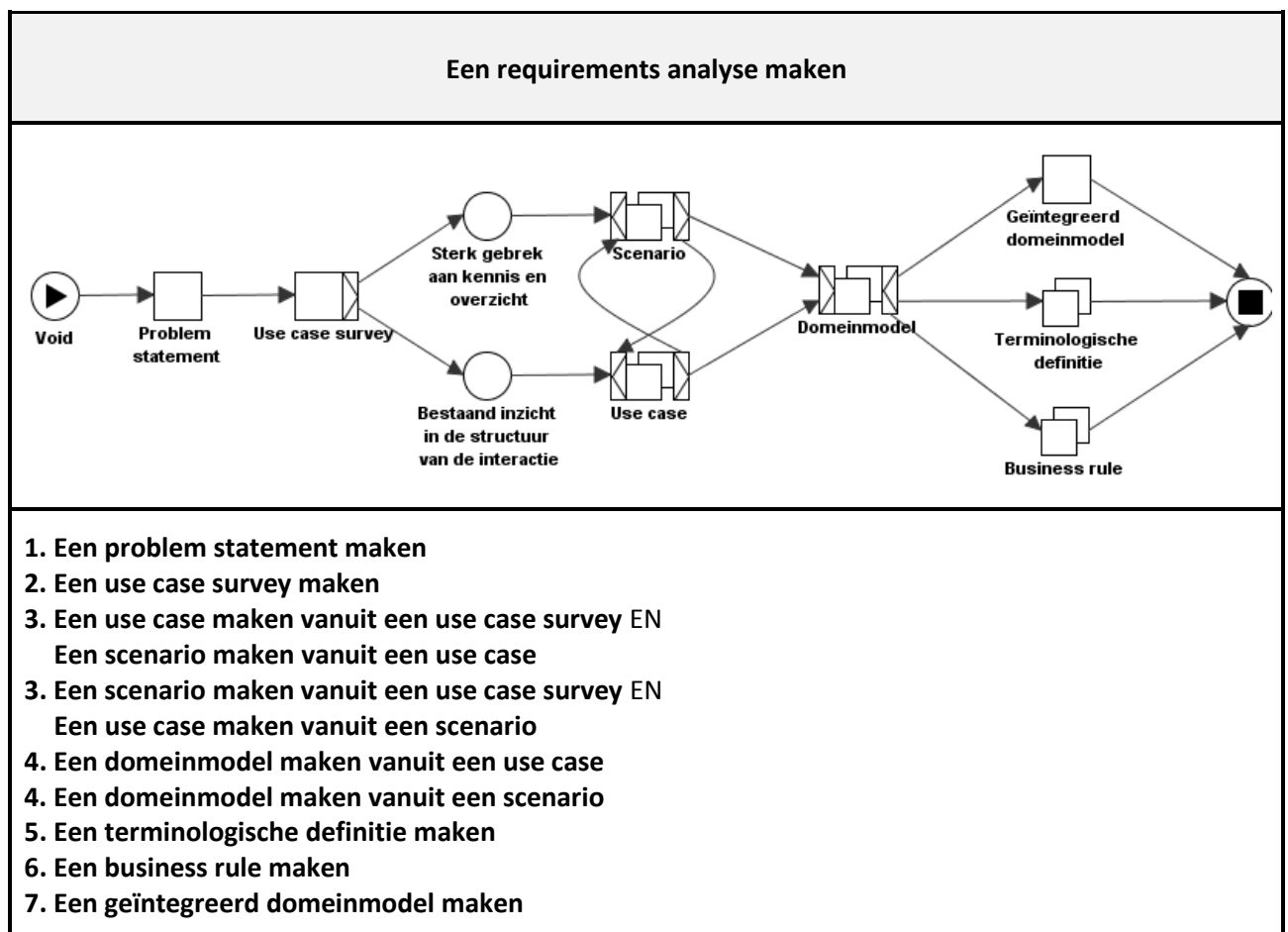
De vierde en laatste deelvraag is de specificatie van een in onze ogen gedegen requirements engineering. Dit alles volgens het raamwerk dat we zelf zojuist hebben gedefinieerd. Het hele stelsel van strategieën kan dienen als een hulpmiddel bij het bedrijven van requirements engineering.

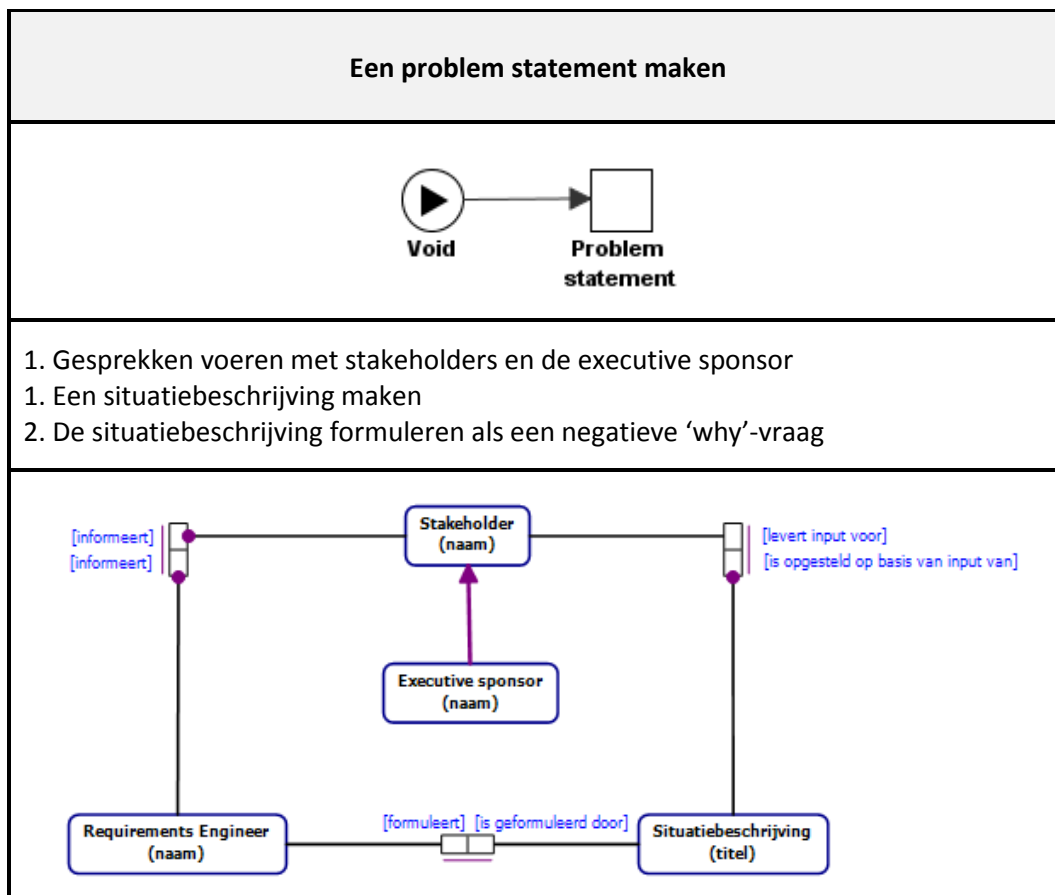
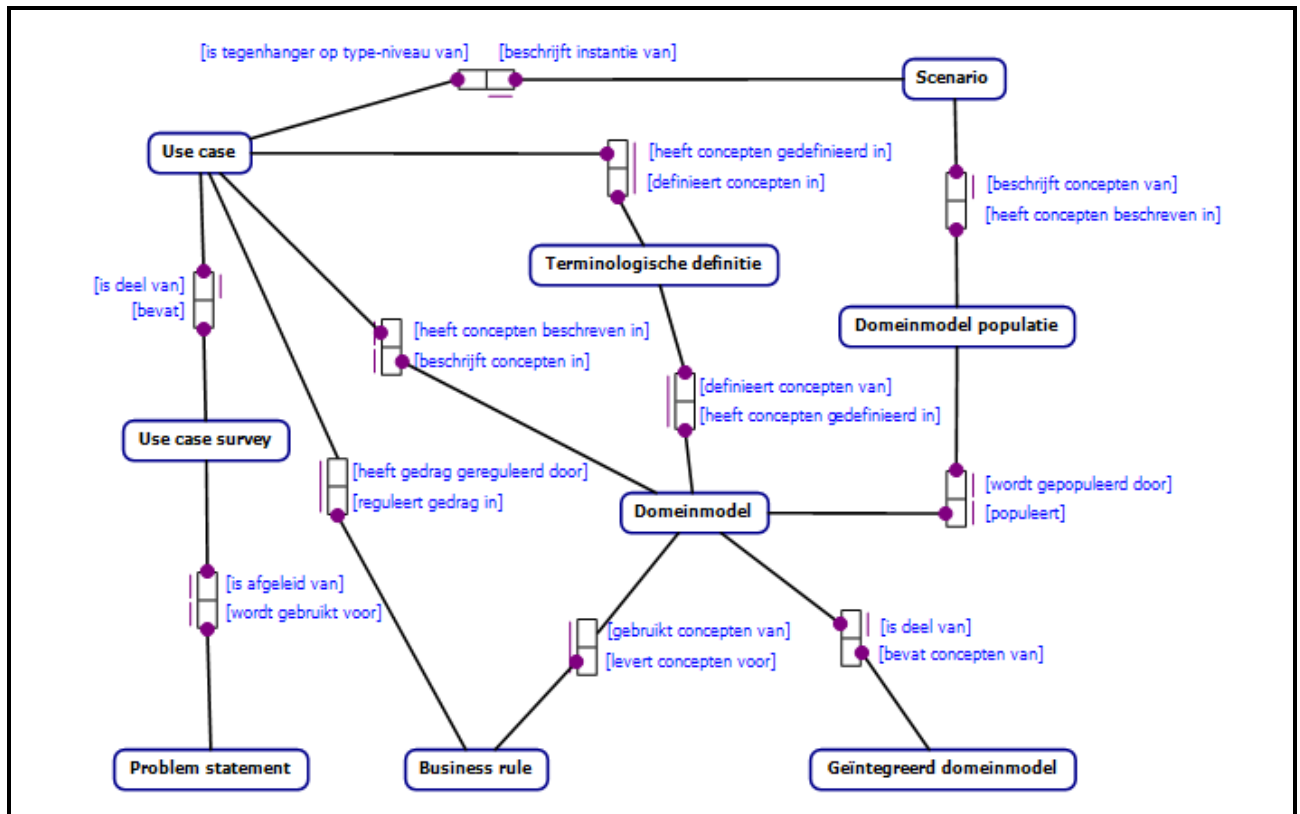
De complete specificatie is ontstaan aan de hand van een aantal workshops met ons en onze begeleider, die tevens de docent van de cursus 'Requirements Engineering' aan onze universiteit is. Dit proces is goed te vergelijken met een modelleersessie waarbij wij de modelleers zijn en de docent de informant en domeinexpert is. Uiteindelijk hebben deze sessies geleid tot een geheel aan activiteiten waarmee het vakgebied van het maken van een requirements analyse goed afgedekt is.

We zullen beginnen met een uitwerking op papier. In sectie 4.2 beschrijven we vervolgens een interactieve online versie van deze specificatie.

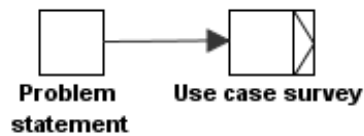
4.1. Uitwerking volgens het raamwerk

Hieronder zullen we alle activiteiten uit de totale specificatie van het concrete proces van requirements engineering op papier weergeven. We beginnen hiermee met de activiteit van het hoogste niveau. Vervolgens gaan we depth-first verder met alle daarin voorkomende samengestelde stappen. De verschillende niveaus geven we hiermee aan via inspringen.





Een use case survey maken



1. Duidelijke afzonderlijke interacties benoemen

- Streef naar compleetheit binnen een bewust bepaalde scope

2. Een lijst van use case namen opstellen

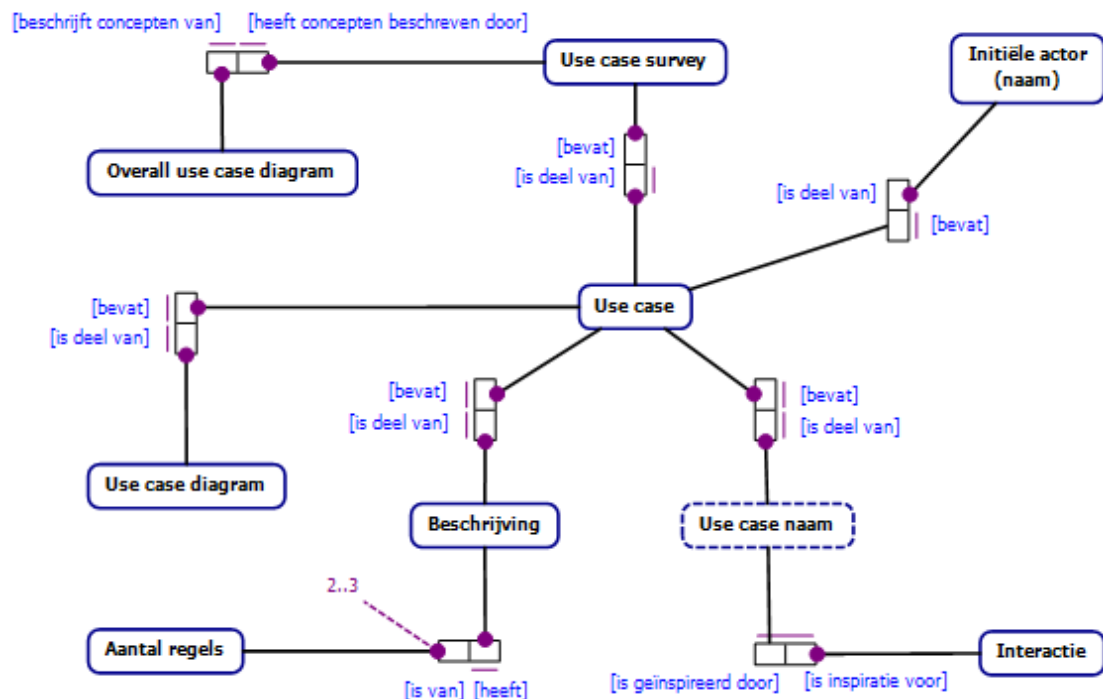
- Gebruik een use case name filter (Kulak & Guiney, 2000, p. 86)

3. Een initiële actor benoemen per use case

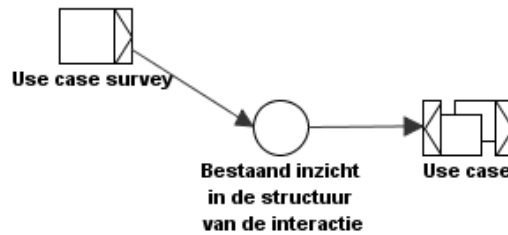
4. Een beschrijving benoemen per use case

- Maak de beschrijving niet langer dan 2-3 regels

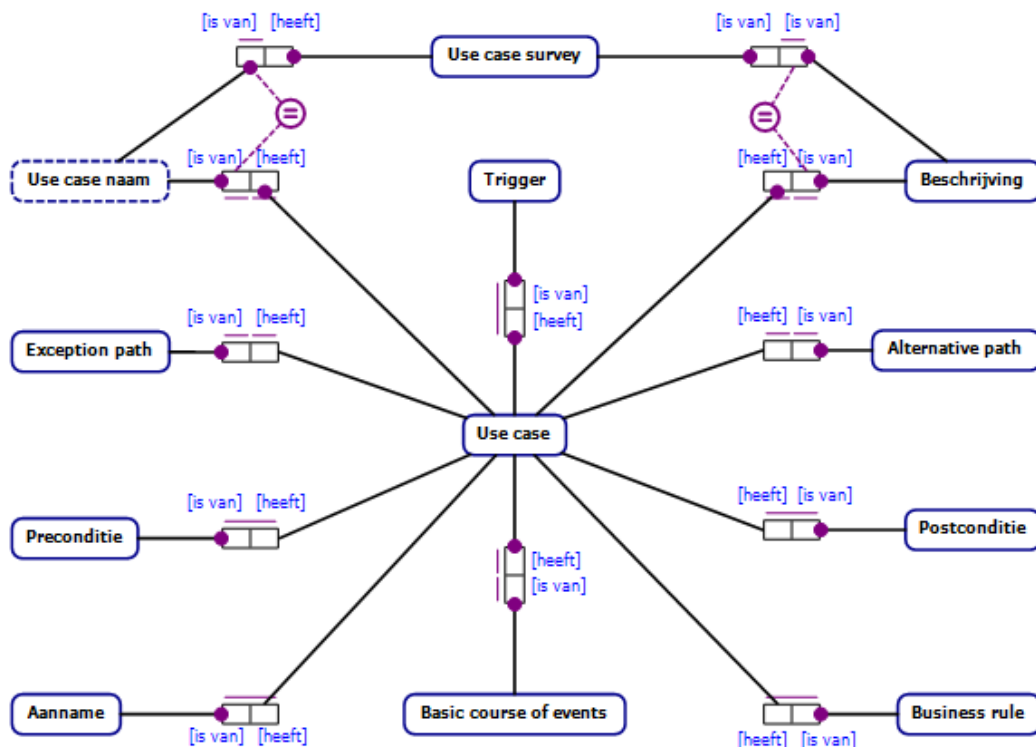
5. Een overall use case diagram maken

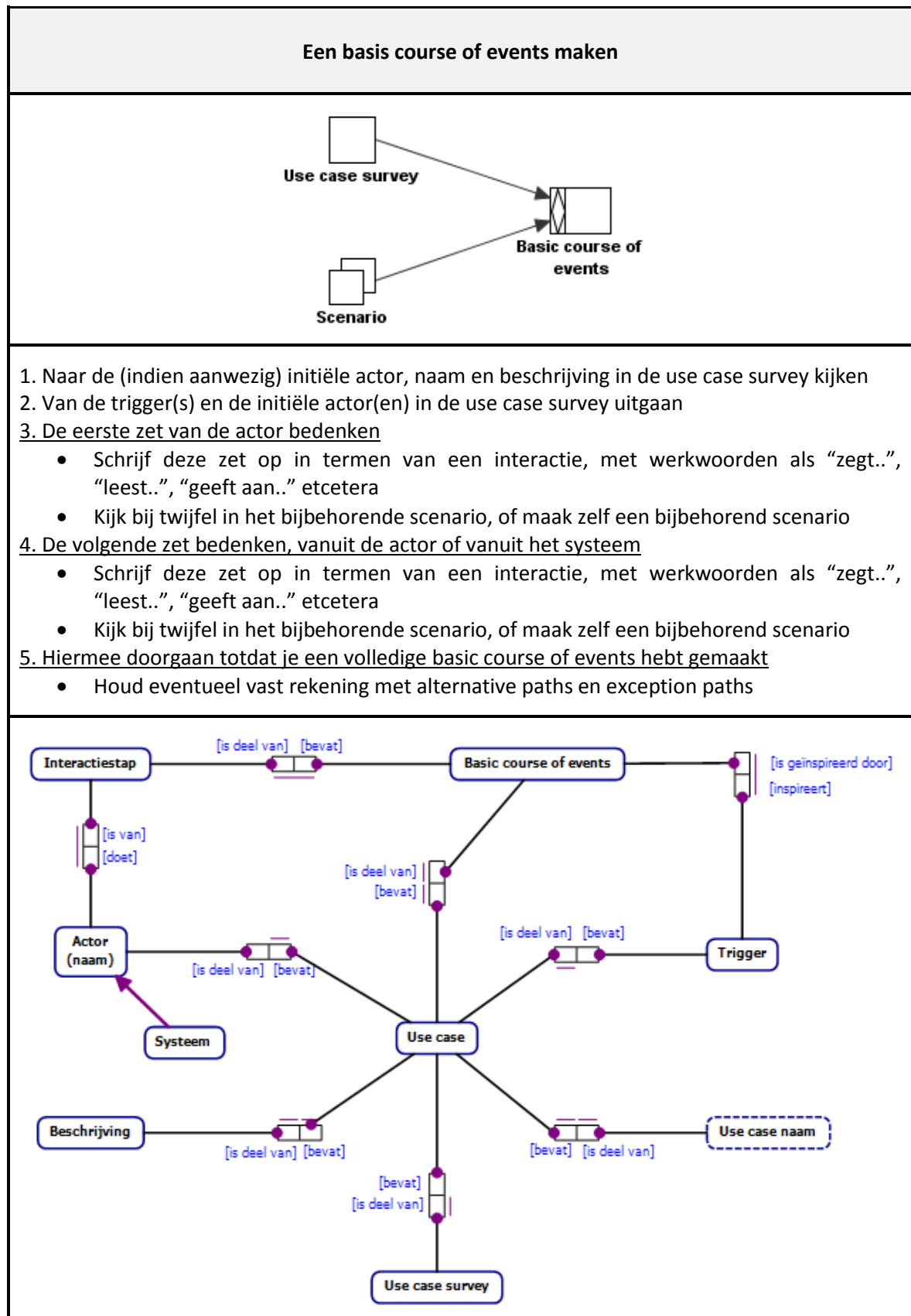


Een use case maken vanuit een use case survey

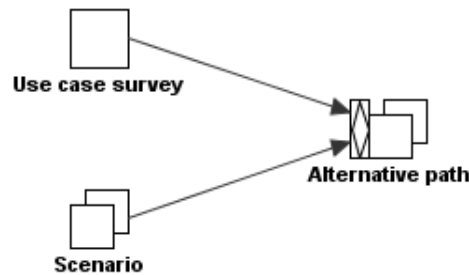


1. De naam en beschrijving van de use case importeren vanuit de use case survey
2. Een basis course of events maken
3. Een alternative path maken
4. Een exception path maken
5. De trigger(s) beschrijven
6. De precondition(s) beschrijven
 - Kijk naar relevante basis courses of events en bepaal aan welke voorwaarden voldaan moet worden voor de interactie uitgevoerd kan worden
 - Besef dat een precondition niet hetzelfde is als een trigger
7. De postconditie(s) beschrijven
 - Kijk naar relevante basis courses of events en bepaal aan welke voorwaarden voldaan moet zijn nadat de interactie uitgevoerd kan worden
8. De aanname(s) beschrijven
9. De gerelateerde business rule(s) noemen
 - Noem enkel business rules die je al hebt, op dit moment hoeft je er zelf nog geen op te stellen

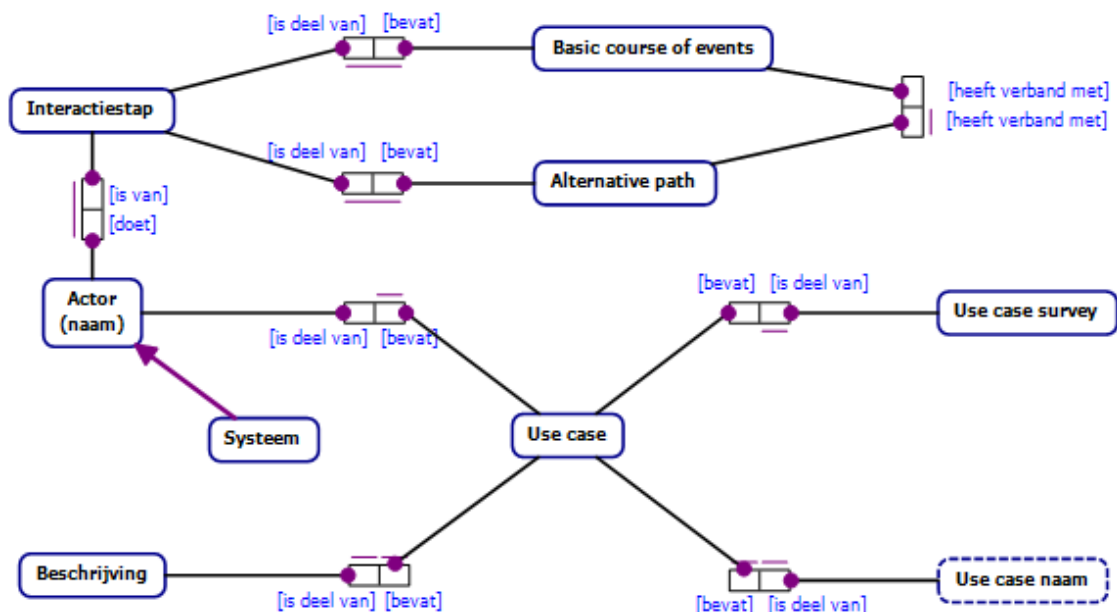




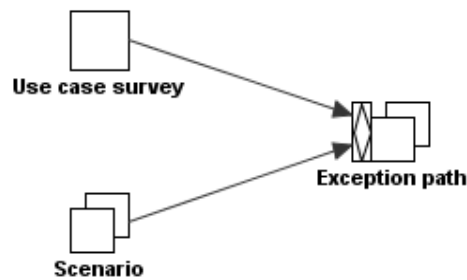
Een alternative path maken



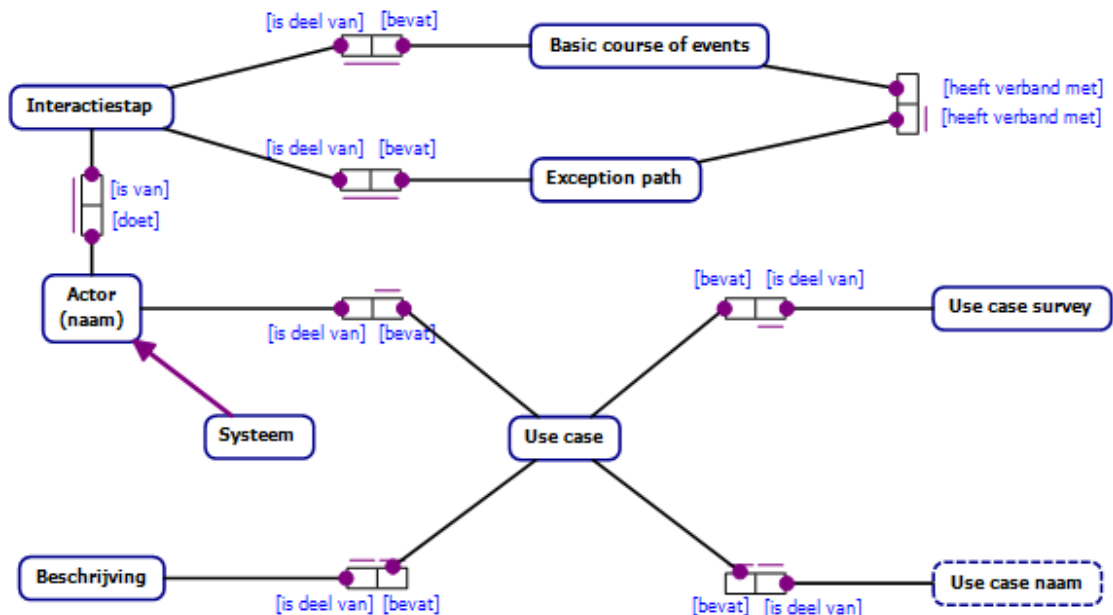
1. Naar de (indien aanwezig) initiële actor, naam en beschrijving in de use case survey kijken
2. Van de niet meest voorkomende en voor de hand liggende gevallen uitgaan, met uitzondering van de gevallen waarin er iets fout gaat
 - Zorg dat er een expliciet en duidelijk verband bestaat tussen de basic course of events en het alternative path
3. De eerste zet van de actor bedenken
 - Schrijf deze zet op in termen van een interactie, met werkwoorden als “zegt..”, “leest..”, “geeft aan..” etcetera
 - Kijk bij twijfel in het bijbehorende scenario, of maak zelf een bijbehorend scenario
4. De volgende zet bedenken, vanuit de actor of vanuit het systeem
 - Schrijf deze zet op in termen van een interactie, met werkwoorden als “zegt..”, “leest..”, “geeft aan..” etcetera
 - Kijk bij twijfel in het bijbehorende scenario, of maak zelf een bijbehorend scenario
5. Hiermee doorgaan totdat je een volledig alternative path hebt gemaakt



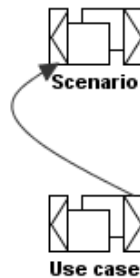
Een exception path maken



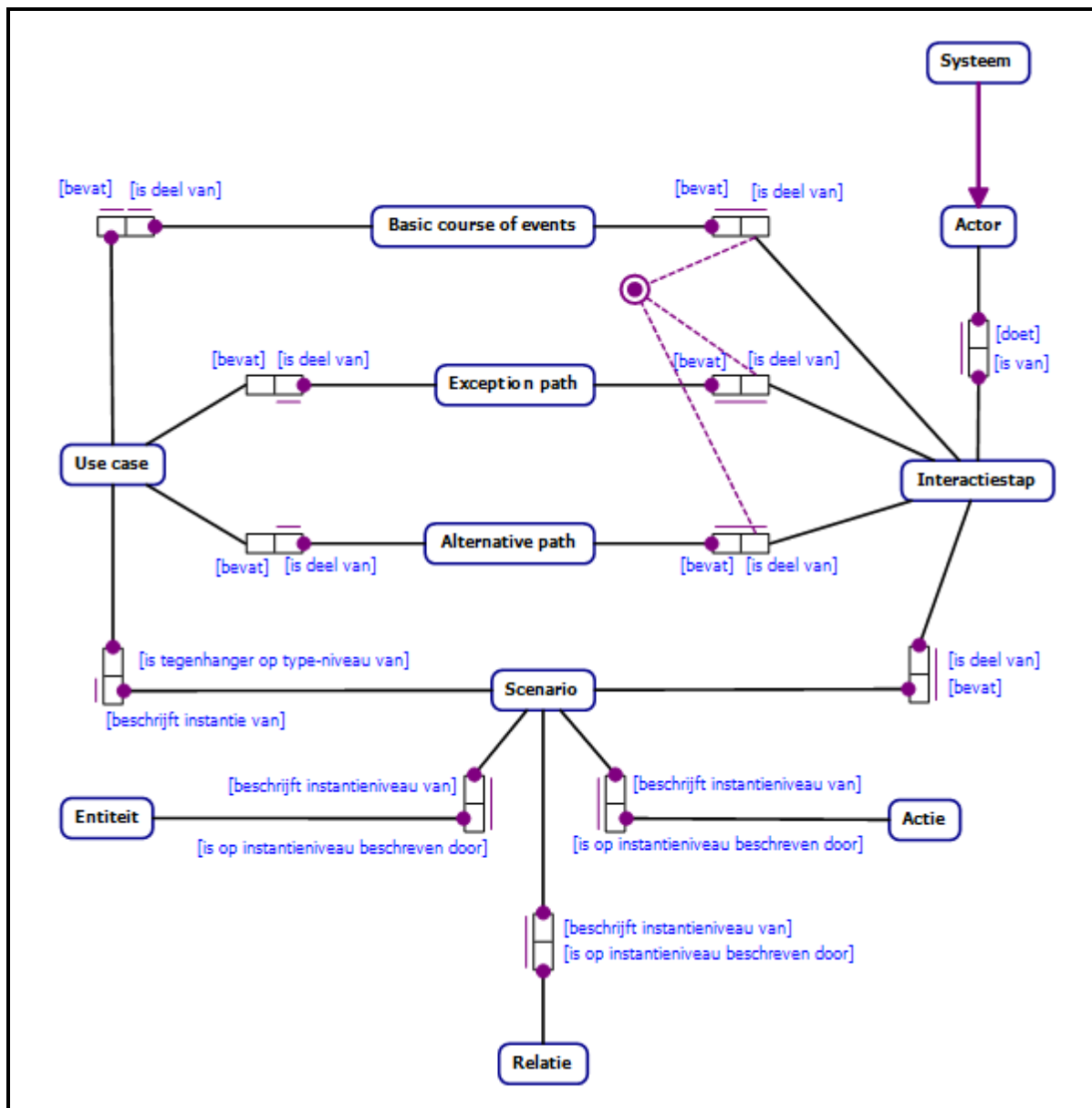
1. Naar de (indien aanwezig) initiële actor, naam en beschrijving in de use case survey kijken
2. Van de gevallen waarin er iets fout gaat uitgaan, met uitzondering van de niet meest voorkomende en voor de hand liggende gevallen
 - Zorg dat er een expliciet en duidelijk verband bestaat tussen de basic course of events en het exception path
3. De eerste zet van de actor bedenken
 - Schrijf deze zet op in termen van een interactie, met werkwoorden als “zegt..”, “leest..”, “geeft aan..” etcetera
 - Kijk bij twijfel in het bijbehorende scenario, of maak zelf een bijbehorend scenario
4. De volgende zet bedenken, vanuit de actor of vanuit het systeem
 - Schrijf deze zet op in termen van een interactie, met werkwoorden als “zegt..”, “leest..”, “geeft aan..” etcetera
 - Kijk bij twijfel in het bijbehorende scenario, of maak zelf een bijbehorend scenario
5. Hiermee doorgaan totdat je een volledig exception path hebt gemaakt



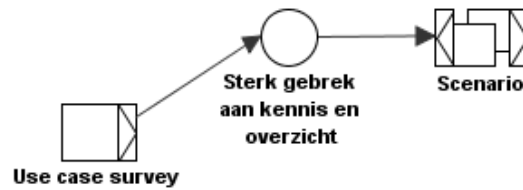
Een scenario maken vanuit een use case



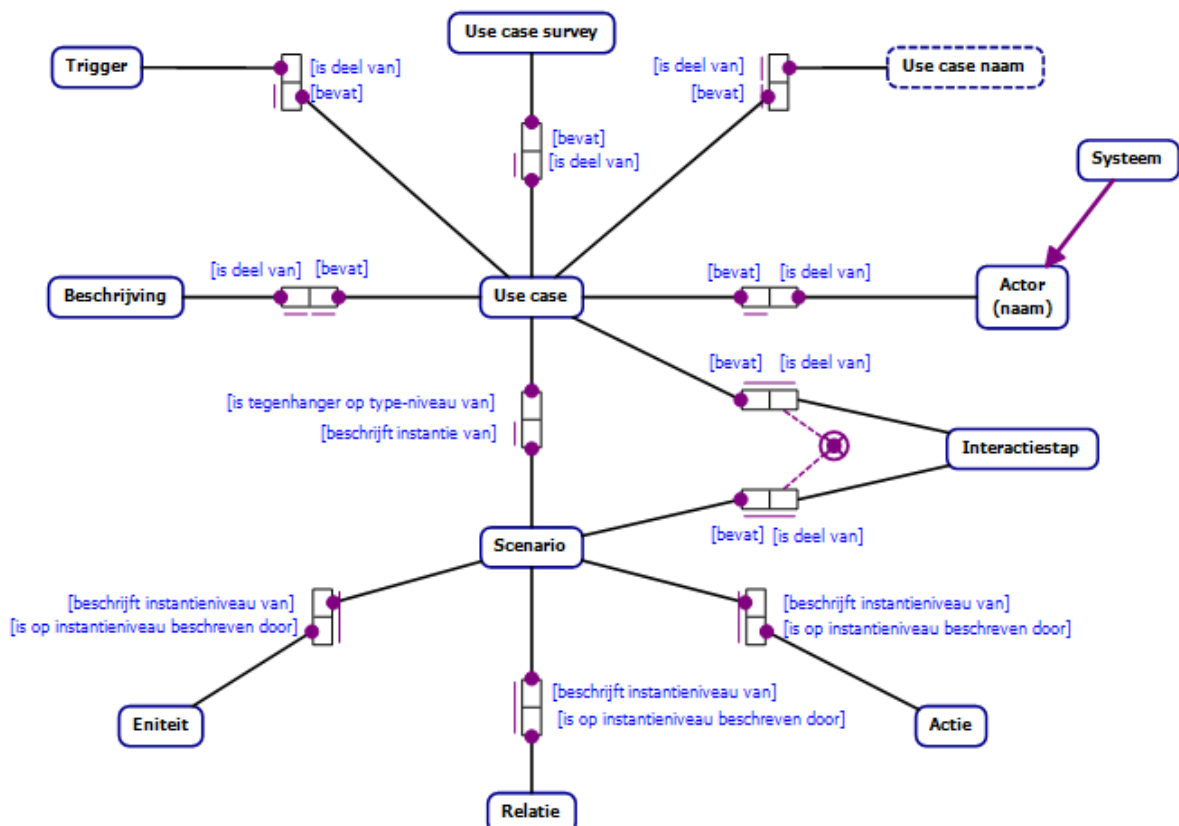
1. Van de basis course of events van de betreffende use case uitgaan
2. Precies vertellen welke interacties er tussen de actor en het systeem plaatsvinden in de volgorde waarin ze plaatsvinden
 - Schrijf dit op in termen van een interactie, met werkwoorden als “zegt..”, “leest..”, “geeft aan..” etcetera
 - Refereer aan de interacties volgens hetzelfde systeem als in de use case
 - Let op dat een scenario instantieniveau beschrijft van alle entiteiten, acties en relaties
3. Het bedenken of er nog andere relevante scenario’s te maken zijn bij deze basic course of events
4. Van de alternative paths van de betreffende use case uitgaan
5. Precies vertellen welke interacties er tussen de actor en het systeem plaatsvinden in de volgorde waarin ze plaatsvinden
 - Schrijf dit op in termen van een interactie, met werkwoorden als “zegt..”, “leest..”, “geeft aan..” etcetera
 - Refereer aan de interacties volgens hetzelfde systeem als in de use case
 - Let op dat een scenario instantieniveau beschrijft van alle entiteiten, acties en relaties
6. Het bedenken of er nog andere relevante scenario’s te maken zijn bij deze alternative paths
7. Van de exception paths van de betreffende use case uitgaan
8. Precies vertellen welke interacties er tussen de actor en het systeem plaatsvinden in de volgorde waarin ze plaatsvinden
 - Schrijf dit op in termen van een interactie, met werkwoorden als “zegt..”, “leest..”, “geeft aan..” etcetera
 - Refereer aan de interacties volgens hetzelfde systeem als in de use case
 - Let op dat een scenario instantieniveau beschrijft van alle entiteiten, acties en relaties
9. Het bedenken of er nog andere relevante scenario’s te maken zijn bij deze exception paths



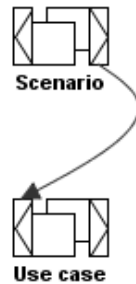
Een scenario maken vanuit een use case survey



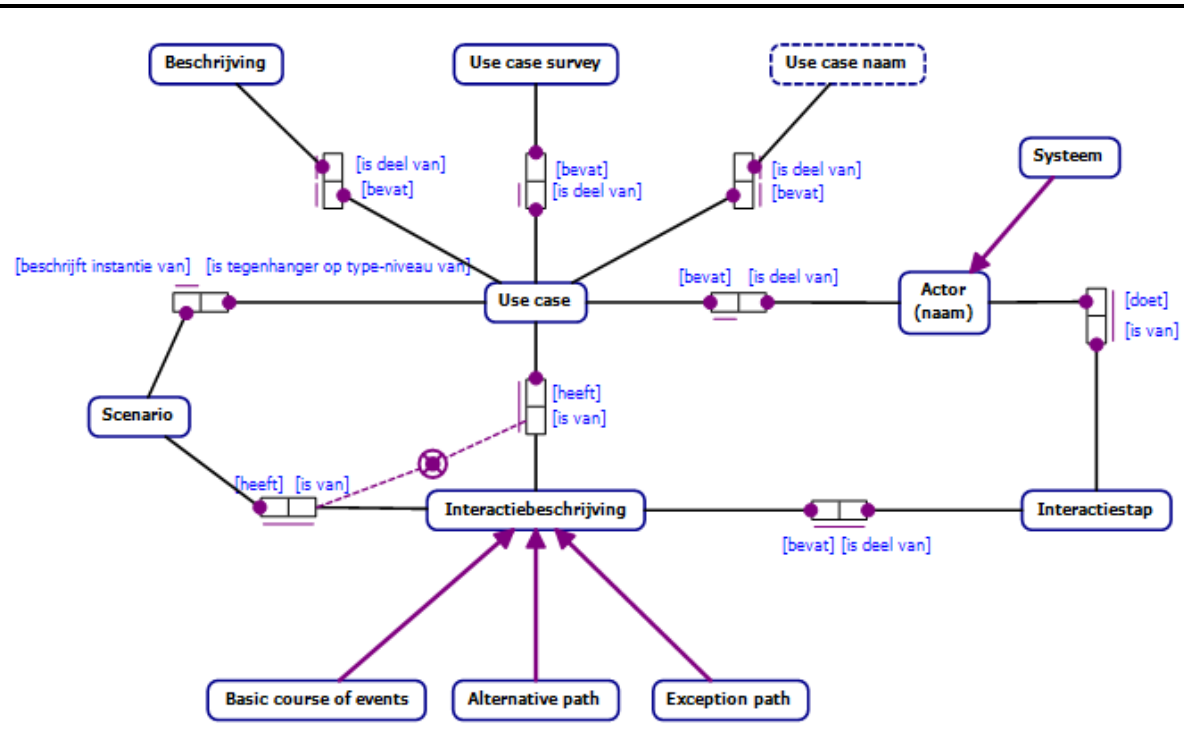
1. Naar de (indien aanwezig) initiële actor, naam en beschrijving in de use case survey kijken
2. Van de trigger(s) en de initiële actor(en) in de use case survey uitgaan
3. Een voorbeeld geven van een specifieke actor (een instantie) die de rol van (initiële) actor zou kunnen vervullen
4. Precies vertellen welke interacties er tussen de actor en het systeem plaatsvinden in de volgorde waarin ze plaatsvinden
 - Schrijf dit op in termen van een interactie, met werkwoorden als “zegt..”, “leest..”, “geeft aan..” etcetera
 - Indien de bijbehorende use case bekend is, refereer dan aan de interacties volgens hetzelfde systeem als in de use case
 - Let op dat een scenario instantieniveau beschrijft van alle entiteiten, acties en relaties
5. Het bedenken of er nog andere relevante scenario's te maken zijn en deze activiteit nogmaals uitvoeren indien dit zo is



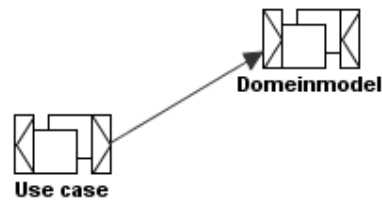
Een use case maken vanuit een scenario



1. De naam en beschrijving van de use case importeren vanuit de use case survey
2. Het scenario lezen en daarbij de stapsgewijze interacties onderscheiden die je aantreft
3. Deze interacties op typeniveau beschrijven en eventueel typenamen verzinnen voor entiteiten, acties en relaties die in het scenario op instantieniveau zijn beschreven
 - Schrijf dit op in termen van een interactie, met werkwoorden als “zegt..”, “leest..”, “geeft aan..” etcetera
4. Het bedenken of deze interactiebeschrijving een basic course of events, alternative path of exception path betreft en ze als zodanig benoemen
5. Eventueel naar het volgende scenario binnen deze use case gaan



Een domeinmodel maken vanuit een use case



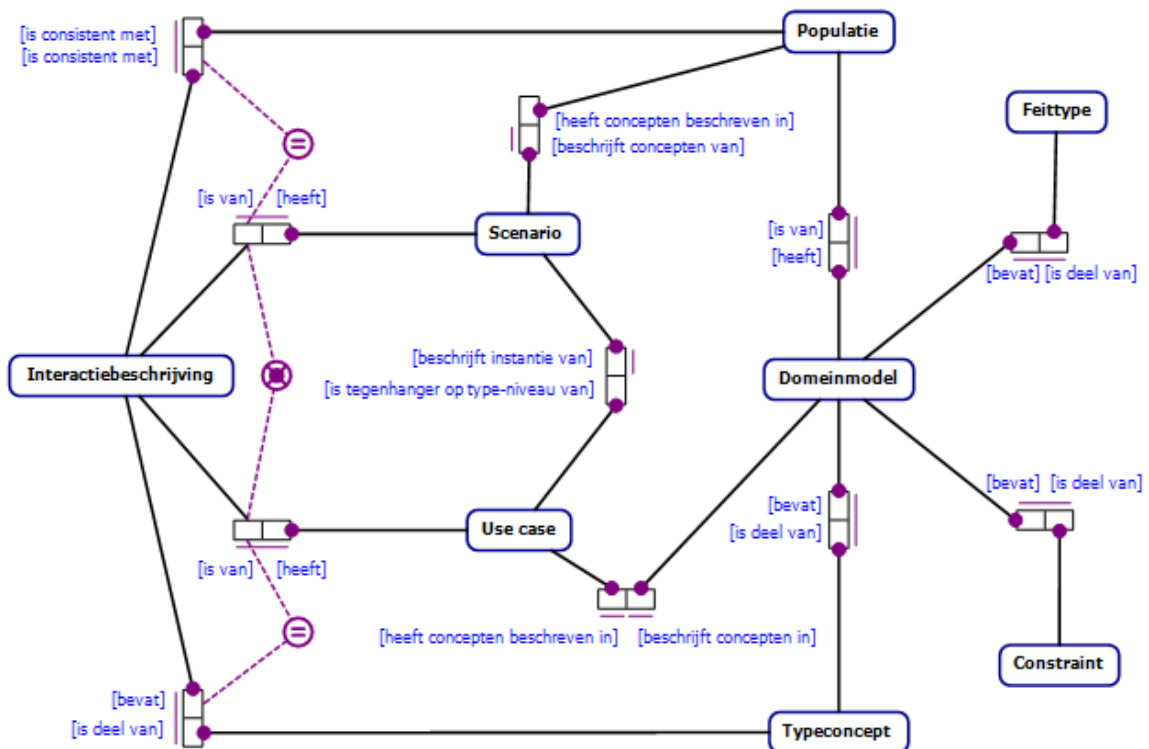
1. Relevante typeconcepten identificeren in een use case

2. De feittypen maken

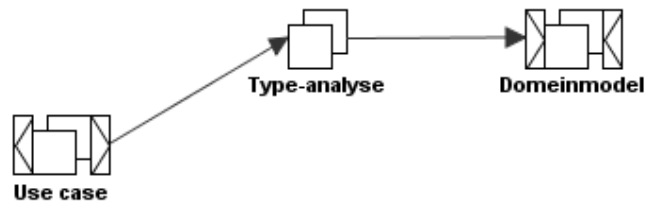
3. Een voorbeeldpopulatie maken

- Doe dit consistent met de scenario's

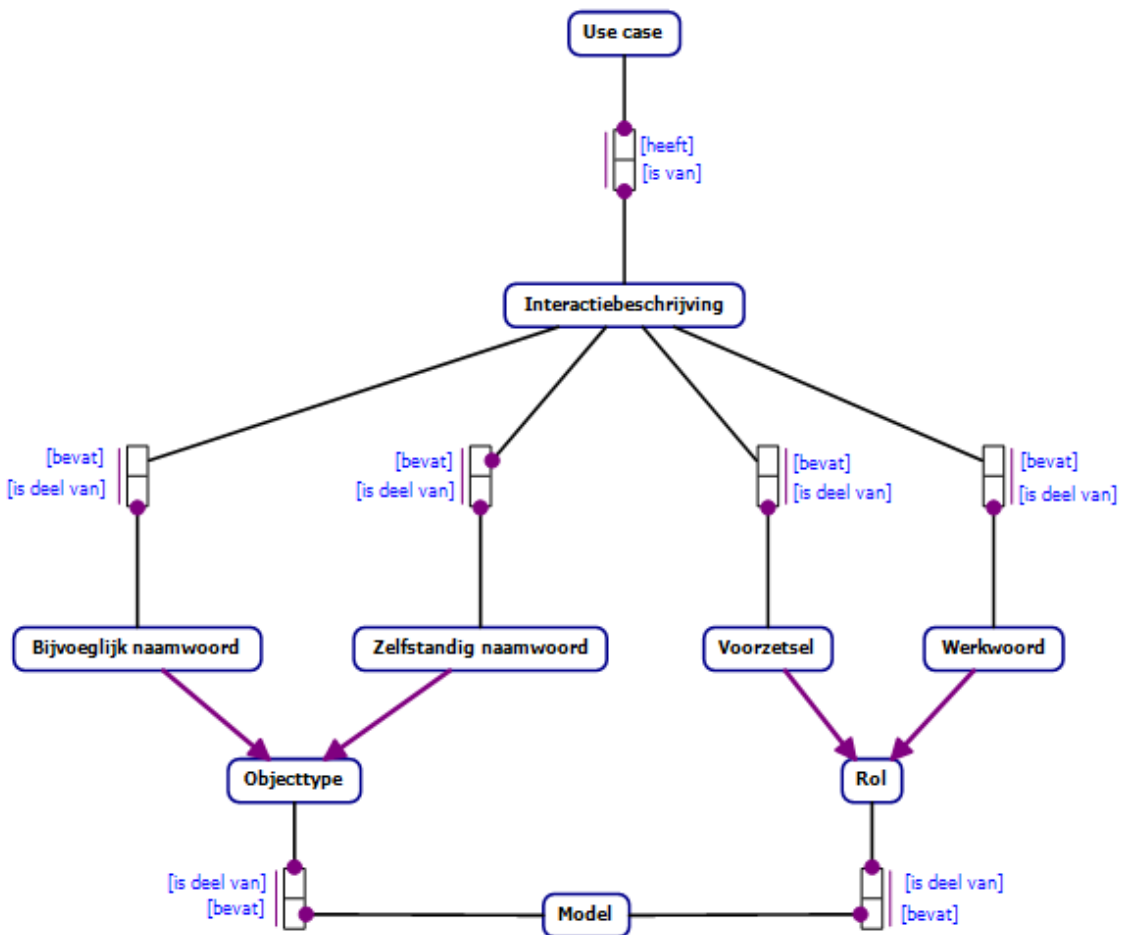
4. De constraints completeren



Relevante typeconcepten identificeren in een use case



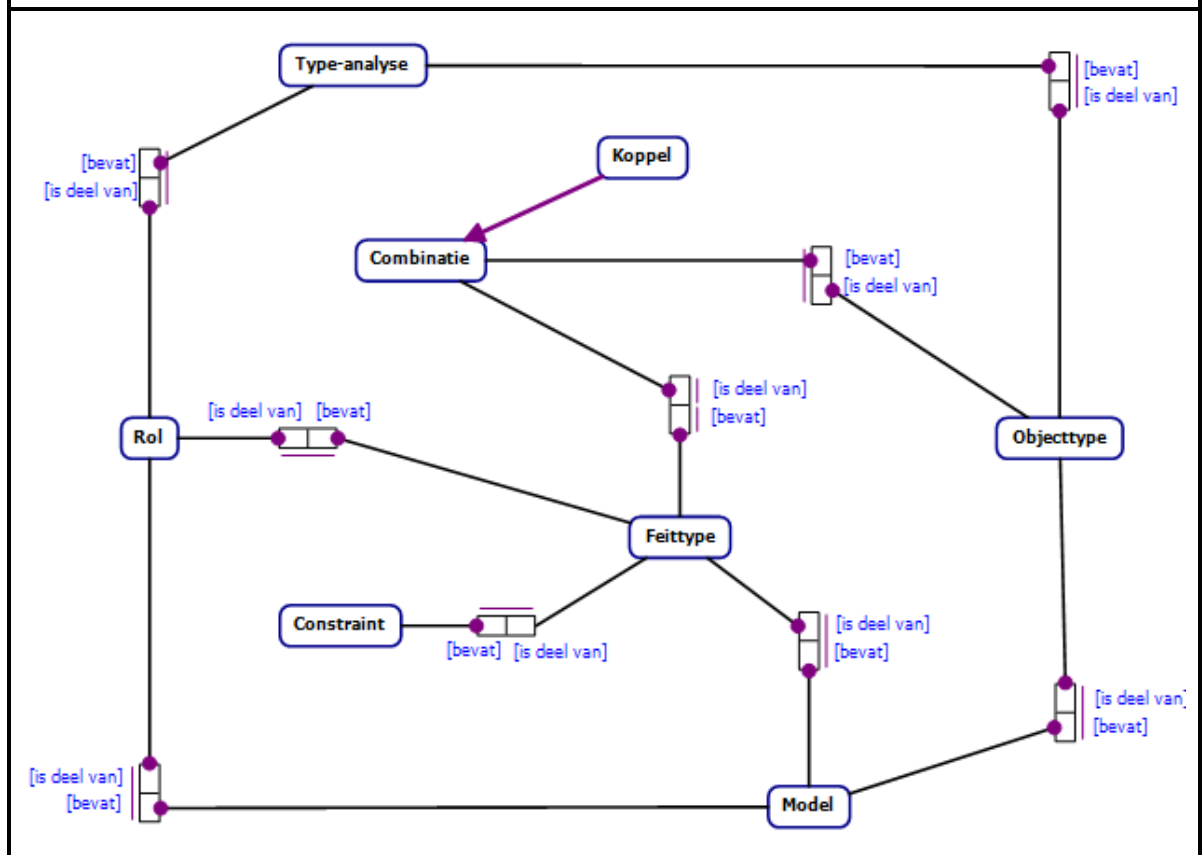
1. Zelfstandige naamwoorden en eventueel bijvoeglijke naamwoorden (de objecttypen) vinden
 - Deze objecten moeten voorkomen in de interactiebeschrijvingen
2. Relevante werkwoorden en voorzetsels (de rollen) vinden
 - Dit zijn typisch interactiewerkwoorden
3. Op verborgen concepten controleren, zoals relevante werkwoorden die er niet helemaal uitzien als interactiewerkwoorden, bijvoorbeeld omdat de use case niet deugt



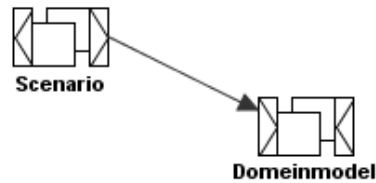
De feittypen maken



1. De koppels van objecttypen enerzijds en rollen anderzijds selecteren
2. De naamgeving van rollen in twee richtingen benoemen
3. De constraints toevoegen waar je ze cruciaal vindt
4. Op mogelijke uitbreidingen controleren van feittypen met twee rollen naar feittypen met meerdere rollen (oftewel het samenvoegen van 'binary facttypes')



Een domeinmodel maken vanuit een scenario



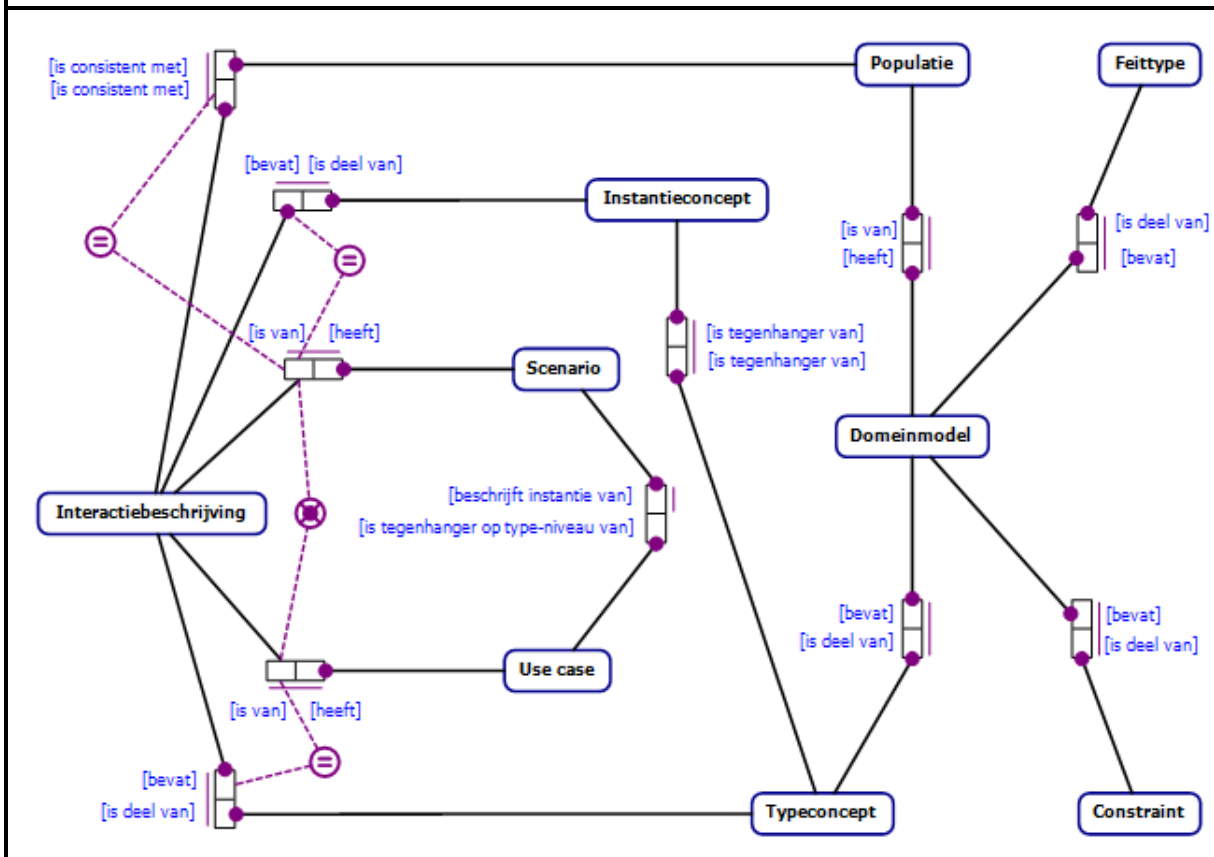
1. Relevante instantieconcepten identificeren in een scenario

2. De feittypen maken

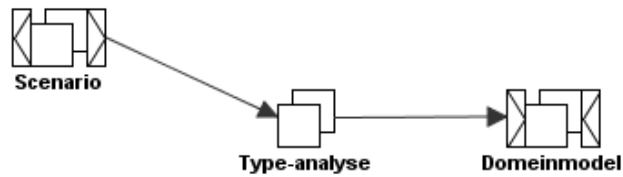
3. Een voorbeeldpopulatie maken

- Doe dit consistent met de scenario's

4. De constraints completeren



Relevante typeconcepten identificeren in een scenario



1. Relevante werkwoorden en voorzetsels (de rollen) vinden

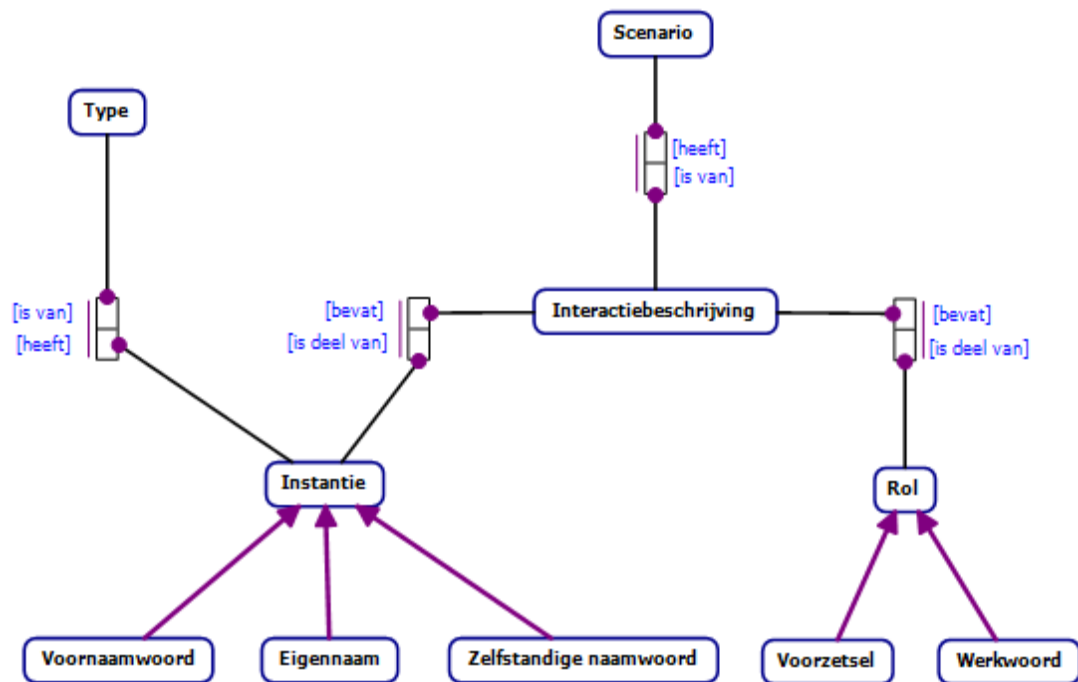
- Dit zijn typisch interactiewerkwoorden

2. Eigennamen, voornaamwoorden en zelfstandige naamwoorden (de instanties) identificeren, die behoren bij de werkwoorden en voorzetsels uit stap 1

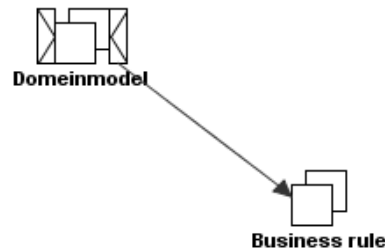
3. De instanties uit stap 2 typeren door te bedenken van welk type ze zijn

- Het gaat hier om de meest specifieke, relevante typering binnen het domein

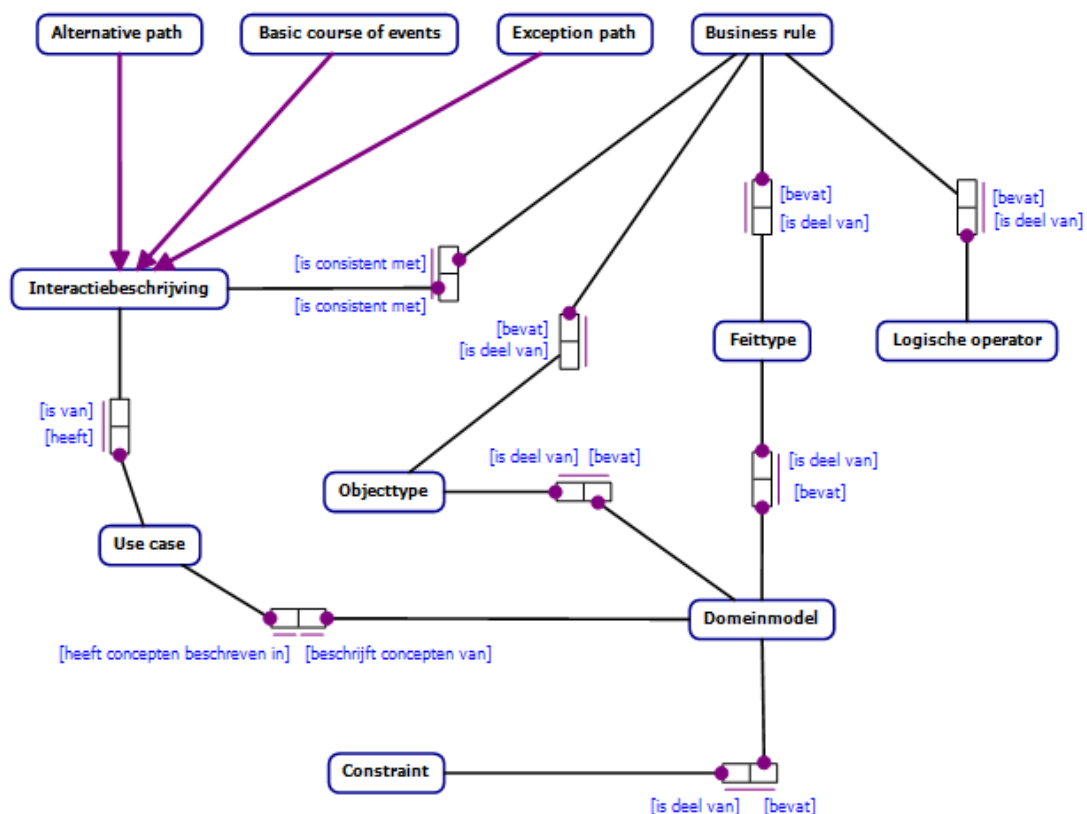
4. Op verborgen concepten controleren, zoals relevante werkwoorden die er niet helemaal uitzien als interactiewerkwoorden, bijvoorbeeld omdat de use case niet deugt



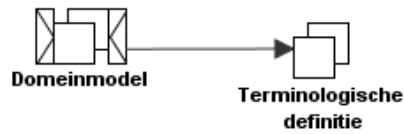
Een business rule maken



1. Een domeinmodel en de use case waaraan deze gekoppeld is erbij pakken
2. Voor elk object en feittype kijken of daar een inperkende regel op van toepassing is
 - Voeg deze regel toe naast bestaande constraints in het domeinmodel
3. Deze regel zo helder mogelijk verwoorden in natuurlijke taal
 - Gebruik voor deze verwoording de verbalisatie van objecttypen en rollen uit het domeinmodel
 - Probeer de formulering in natuurlijke taal zo veel mogelijk de structuur mee te geven van declaratieve expressies met logische operatoren
4. Objecttypen of feittypen toevoegen aan het domeinmodel, indien deze nodig zijn voor de uitdrukking van de regel en nog niet in het domeinmodel voorkomen
5. Ervoor zorgen dat de paden in de use cases (bij zowel de 'basic course of events' als de 'alternative paths' en de 'exception paths') overeenkomen met de relevante business rule, indien de regel consequenties heeft voor temporele structuren (processtructuren)
 - Vaak zal een schending van een business rule leiden tot exceptions



Een terminologische definitie maken



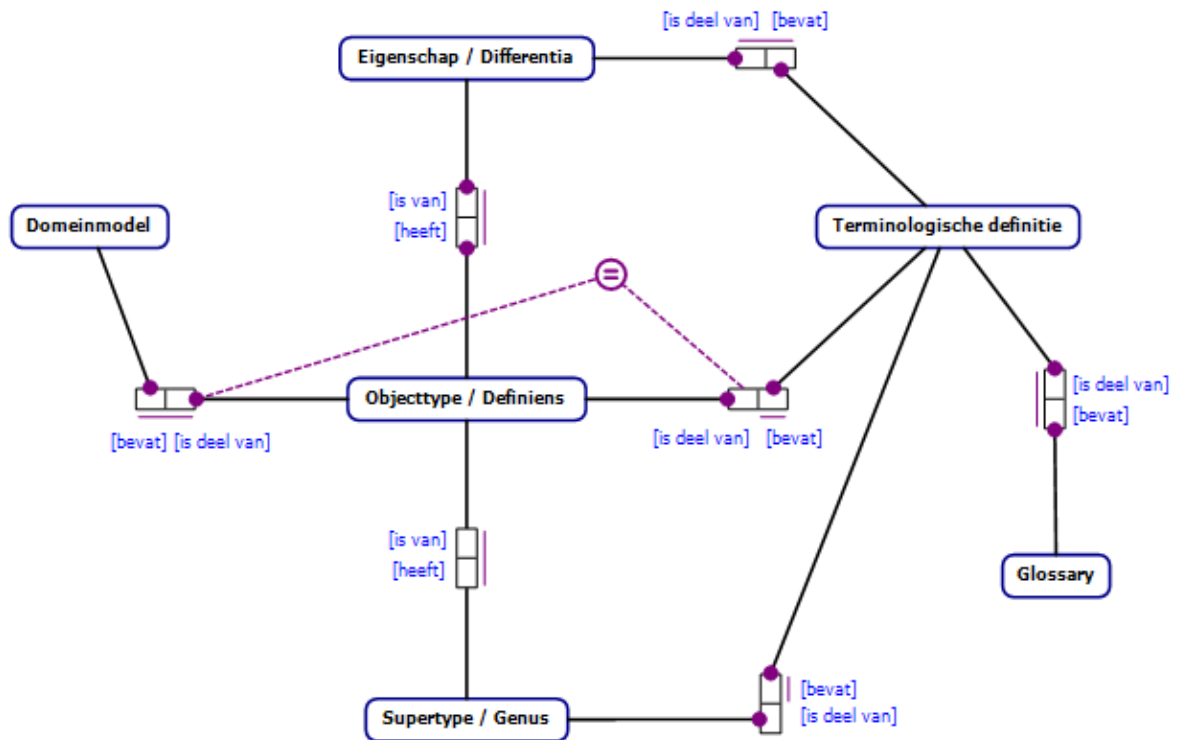
1. Naar een objecttype ('definiens') in je domeinmodel kijken en eventueel naar de use cases en business rules waar deze in voorkomt

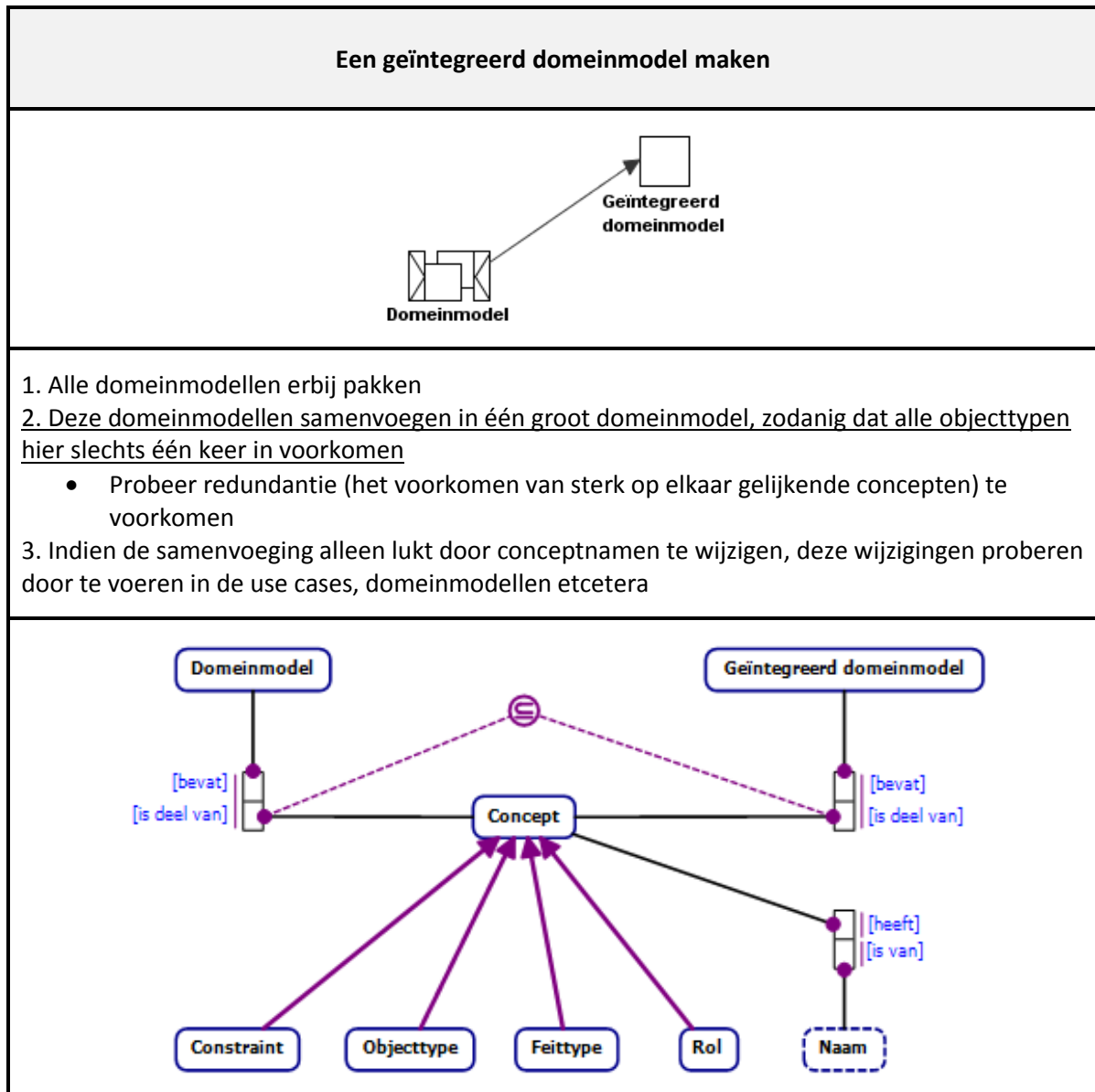
2. Het supertype ('genus') bedenken waar het objecttype onder valt (de 'is-a'-relatie)

- Zorg ervoor dat dit type relevant en zo specifiek mogelijk is

3. Eén of meer eigenschappen bedenken die dit objecttype onderscheiden van andere objecttypen binnen dit supertype

- Zorg ervoor dat de onderscheidende eigenschappen ('differentia') echt bruikbaar zijn voor het maken van onderscheid
- Probeer je te richten op essentiële eigenschappen





4.2. Interactieve uitwerking middels een database

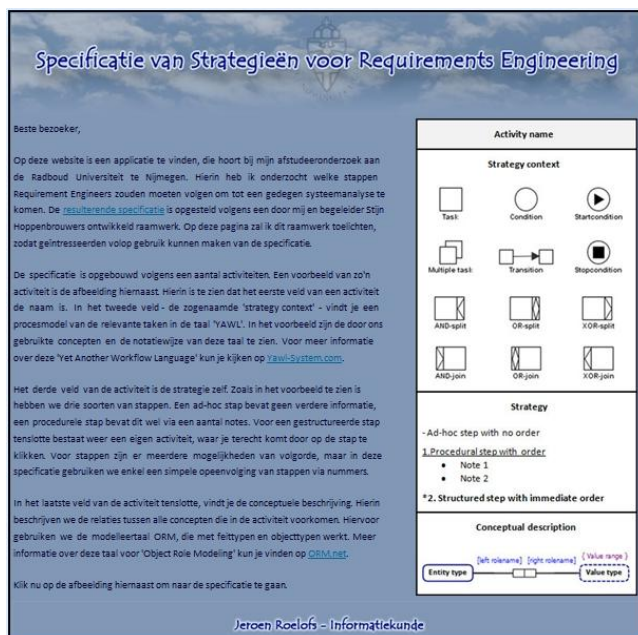
In de vorige sectie zijn alle activiteiten uit de specificatie van ons concrete proces van requirements engineering de revue gepasseerd. Tijdens het creëren van deze activiteiten kwamen we snel tot de conclusie dat het handig zou zijn om ook een soort van elektronische versie van de specificatie in elkaar te zetten. Dit had meerdere voordelen boven de papieren specificatie.

Allereerst is het zo dat we met een interactieve, elektronische uitwerking meer technische snufjes toe kunnen passen. In de papieren versie is het overzicht soms moeilijk te behouden. Je kunt niet in één oogopslag zien waar de activiteiten nu staan die volgen uit bepaalde samengestelde stappen. In de interactieve uitwerking kunnen we van samengestelde stappen hyperlinks maken, waardoor je met één muisklik direct vanaf een samengestelde stap bij de hieruit volgende activiteit terecht komt. Ook kunnen we bij elke activiteit automatisch laten aangeven wat de bovenliggende activiteiten zijn, zodat de structuur te allen tijde duidelijk blijft. Dit brengt de specificatie echt tot leven!

Voor het laten werken van deze online specificatie, dienden we gebruik te maken van een database. Ook dat is als een voordeel te beschouwen. Op deze manier dwingen we onszelf namelijk om alle activiteiten en de bijbehorende velden daarvan in het databasekeurslijf te persen. De structuur hiervan ligt vast, waardoor oneffenheden sneller aan het licht komen. Zo is het niet mogelijk om een samengestelde stap te maken zonder dat er een activiteit bestaat met dezelfde naam. Op deze wijze zijn er meerdere punten boven water gekomen die we wellicht niet hadden ontdekt zonder deze elektronische versie.

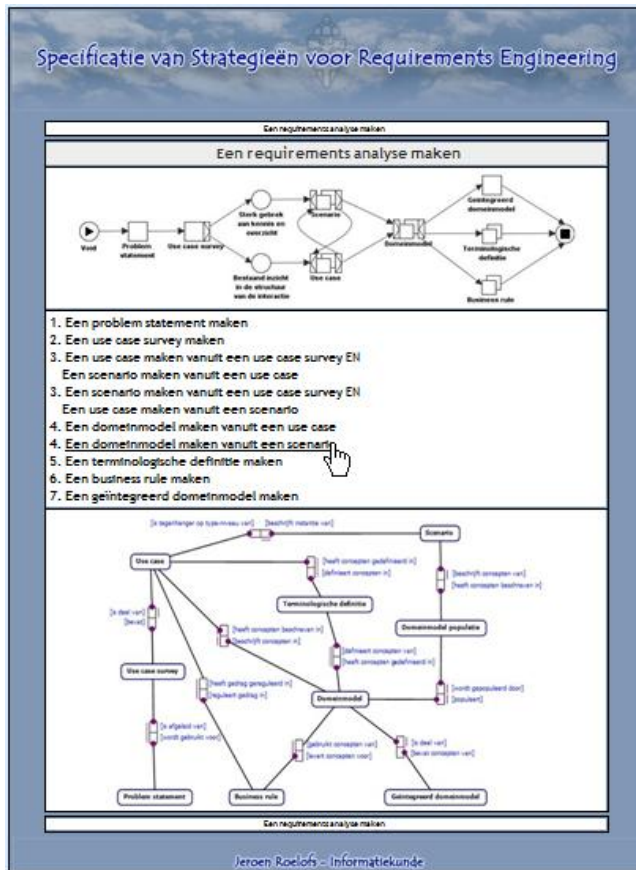
In deze masterscriptie willen we niet alleen verwijzen naar de pagina waarop deze online applicatie te vinden is. Allereerst hebben we in de eerste bijlage van dit document (zie sectie 8.1) beschreven hoe de database eruit ziet. Hierbij zijn de verschillende tabellen en de velden daarvan uiteengezet en is van elk veld beschreven waar het toe dient. In de tweede bijlage (zie sectie 8.2) is een deel van de programmatuur van de online applicatie te vinden. Het gaat hier om het deel waarmee de activiteiten uit de database gehaald worden. Maar naast deze bijlagen over de opbouw en structuur van de interactieve werking, laten we in het vervolg van deze sectie de lezer ook een aantal screenshots zien.

De online applicatie is te vinden via het adres '<http://www.rhizon.nl/reqspec>'. Op de eerste pagina waar je dan terecht komt, staat nogmaals in het kort uitgelegd hoe het raamwerk ook weer werkte. Tevens wordt er voor de *strategy context* en de *conceptual description* verwezen naar respectievelijk de website yawl-system.com en de website orm.net. Een screenshot van deze eerste pagina is te zien in figuur 17.



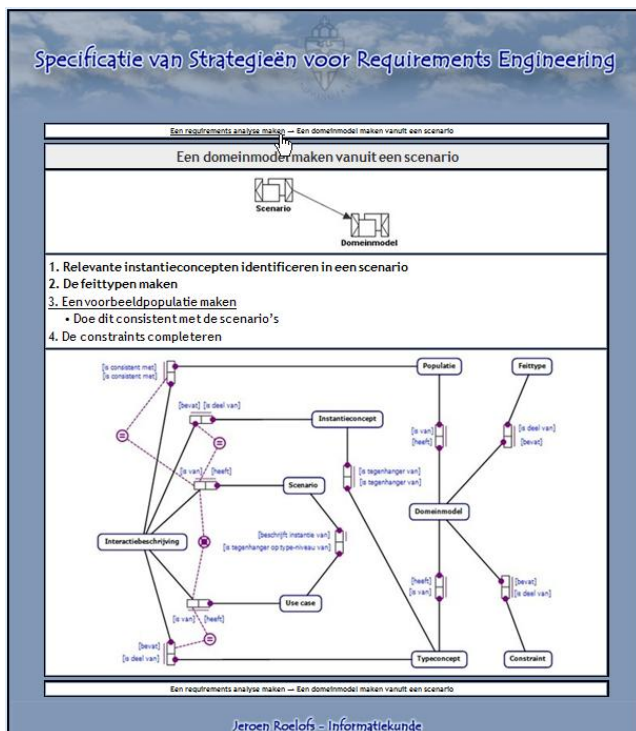
Figuur 17: voorpagina van online applicatie

Wanneer je vanaf deze pagina verder gaat, kom je terecht bij de eerste activiteit. Deze activiteit – ‘Een requirements analyse maken’ – zie je in figuur 18. Op deze afbeelding is precies dezelfde activiteit te zien zoals in de vorige sectie van dit hoofdstuk. Er is echter ook te zien dat de muiscursor zich boven de samengestelde stap ‘Een domeinmodel maken vanuit een scenario’ bevindt. Hierbij komt het accent op deze samengestelde stap te liggen en is tevens aan de muiscursor te zien dat het hier een link betreft. Wanneer de gebruiker op deze link klikt, komt hij of zij direct terecht bij de pagina waarop de ingezoomde activiteit staat waarin beschreven staat hoe je vanuit een domeinmodel een scenario dient te maken. Zo kun je door de hele specificatie heen navigeren.



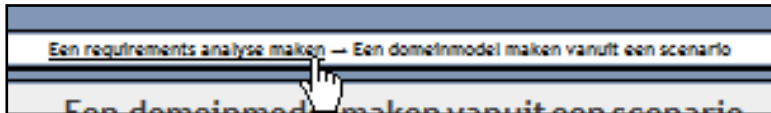
Figuur 18: activiteit binnen online applicatie

Figuur 19 hieronder toont nog een screenshot waarin de pagina te vinden is waar je terecht komt na het klikken op de link.



Figuur 19: activiteit na volgen samengestelde stap binnen online applicatie

Naast de activiteiten is direct boven en onder ook nog een zogenaamde 'locatiebalk' te zien. Hierin wordt de plaats van de huidige activiteit in de structuur van de specificatie getoond. In figuur 20 is een ingezoomde afbeelding van deze locatiebalk te vinden. Hierin is direct te zien dat de activiteit 'Een domeinmodel maken vanuit een scenario' volgt uit een samengestelde stap in de activiteit 'Een requirements analyse maken'. Tevens is het mogelijk om direct naar deze bovenliggende activiteit toe te gaan door op deze link te klikken.



Figuur 20: locatiebalk online applicatie

Hiermee hebben we genoeg gezegd over de online uitwerking van de specificatie. Zoals eerder gezegd is deze applicatie te vinden op de website '<http://www.rhizon.nl/reqspec>'.

5. Conclusie en toekomstig werk

In dit hoofdstuk zullen we een aantal conclusies trekken over de deelvragen die de aanleiding vormden voor dit afstudeeronderzoek. Daarnaast zullen we zelf een aantal discussiepunten opwerpen over onze aanpak en het uiteindelijke resultaat daarvan. Tenslotte zullen we wat onderwerpen aankaarten waarmee ander onderzoek verder kan gaan waar deze scriptie ophoudt.

De eerste deelvraag betrof concrete voorbeeld waarmee we het begrip 'requirements engineering' konden operationaliseren. We hebben ervoor gekozen om dit in te vullen volgens de werkwijzen zoals die in de cursus 'requirements engineering' op onze universiteit gedoceerd wordt. Het is altijd moeilijk om een zo goed mogelijke versie van dit vakgebied te kiezen, aangezien het op veel verschillende manieren bedreven wordt. Wat de één goed vindt werken, zal een ander nooit willen gebruiken. Met andere woorden, er is geen *silver bullet* voor requirements engineering.

Daarom kunnen we ons best voorstellen dat mensen het niet eens zullen zijn met onze strategieën, maar dat nemen we dan maar voor lief. Het is voor ons juist belangrijk dat we een zo goed mogelijk inzicht hebben in de manier waarop de cursus het vakgebied voorschrijft. De informant en docent van de cursus heeft meerdere jaren ervaring in het doceren ervan en weet daarom waar de valkuilen zitten. Die ervaring is eigenlijk essentieel geweest voor het opstellen van de specificatie. Zo wisten we precies welke stappen we wel en niet moesten noemen om het voor gebruikers begrijpbaar te maken. Daarom is het kiezen van de cursus 'requirements engineering' voor de operationalisatie van dit afstudeeronderzoek de juiste keuze geweest.

Met de tweede deelvraag komen we terecht bij het raamwerk dat we in dit afstudeeronderzoek opgebouwd hebben. Zoals gezegd was het oorspronkelijke raamwerk van de onderzoeksgroep IRIS (Van Bommel, 2007) slechts een eerste aanzet. Eén van de doelen van ons onderzoek was juist om het raamwerk aan te passen door het te in de praktijk te gebruiken. Het oorspronkelijke raamwerk bleek niet echt goed te werken. Tijdens de modelleersessies waarbij de specificatie opgebouwd werd, liepen we meerdere keren tegen problemen aan. Zo konden we via een graphstructuur geen complexere *splits* en *joins* definiëren. Ook was het moeilijk om de condities van bepaalde keuzes in het proces weer te geven en konden we niet aangeven dat een activiteit meerdere keren uitgevoerd dient te worden. Een kort kijkje in de keuken van YAWL leerde ons dat deze modelleertaal al deze constructies wél in zich had. De keuze om graphen in te ruilen voor procesmodellen in YAWL was dus snel gemaakt. Ook YAWL is een erg intuïtieve taal en sinds de overstap hebben we over de *strategy context* van de activiteiten nooit meer iets te klagen gehad. Een goede keuze dus!

Hiernaast hebben we de beschrijving van het oorspronkelijke raamwerk een stuk concreter gemaakt en hebben we her en der nog wat extra constructies in moeten bouwen om het allemaal in de praktijk te laten werken. Ook hebben we een wat andere naamgeving gebruikt. Aangezien de specificatie zelf volgens ons een succesverhaal is geworden, zou dit ook voor het raamwerk moeten gelden. We kunnen op dit moment echter niet de conclusie trekken dat ons raamwerk in alle gevallen erg bruikbaar is. We hebben het namelijk enkel getest op onze operationalisatie van het vakgebied 'requirements engineering'. Dat het hiervoor werkt is niet voldoende bewijs dat het raamwerk ook bruikbaar is voor heel andere takken van sport. Het gebruiken van het raamwerk in meer diverse situaties is daarom een richting die eventueel 'toekomstig werk' in zou kunnen gaan.

De overeenkomsten en verschillen tussen ons raamwerk en andere soortgelijke raamwerken betreft de derde deelvraag. Hierin hebben we achteraf bekeken in hoeverre ons onderzoek lijkt op de 'state of the art' in de wereld van de method engineering. We kunnen ons indenken dat het feit dat we dit literatuuronderzoek pas na afloop van ons onderzoek uitgevoerd hebben enige discussie oproept. Je zou namelijk kunnen beweren dat we hierdoor het wiel opnieuw uitgevonden hebben. Dat kost veel tijd en zou verspilde moeite kunnen zijn. Dit hebben we echter zelf ook terdege beseft. We vonden het interessanter om een consistent raamwerk op te bouwen volgens de ideeën die we daar zelf over hadden. Zo zouden we ook op wat nieuwe dingen uit kunnen komen, die we wellicht over het hoofd hadden gezien als we ons raamwerk direct hadden gebaseerd op wat er al voorhanden was.

In dit literatuuronderzoek hebben we gezien dat veel raamwerken – waaronder ook de onze – veel soortgelijke elementen bevatten. Bijna elk raamwerk bevat wel een conceptuele beschrijving en een zekere vorm van een procesmodel. Wat dat betreft zou je dus kunnen zeggen dat dit onderzoek inderdaad vooral het wiel opnieuw uitvindt. Het is echter blijkbaar gewoon zo dat je onder deze elementen van het raamwerk niet uit kunt komen. De tweede conclusie uit het literatuuronderzoek is namelijk dat de manier waarop wij het ingevuld hebben wel degelijk vernieuwend is. We zijn heel complexe raamwerken tegen gekomen, terwijl we zelf toch een vrij overzichtelijk en begrijpbaar raamwerk hebben opgebouwd. Ook hebben we veel modelmatige alternatieven voor onze tekstuele strategiestappen gezien. Dat heeft enerzijds het voordeel dat je er geen constructies voor bijvoorbeeld de volgorde van de stappen aan toe hoeft te voegen. Het nadeel is dat je vast zit aan een modelleertaal. Met name de manier waarop wij via onze samengestelde stappen verschillende lagen implementeren zie je niet veel terug, terwijl het wel erg goed werkt. In ieder geval menen we dat ons raamwerk ook 'state of the art' genoemd mag worden.

De laatste deelvraag betrof de specificatie zelf, die we kunnen beantwoorden door naar hoofdstuk vier te verwijzen. Wat wel een punt van discussie zou kunnen zijn, is de mate van detail die we in de beschrijving gekozen hebben. Zo hebben we ervoor gekozen om iteraties (zoals de verschillende fasen van producten) eruit te laten en bepaalde stappen als ad-hoc te definiëren. Deze beslissingen zijn echter gestoeld op de ervaring van onze informant en docent van de cursus. Voor de rest staat de specificatie op zichzelf en is het een zeer complete en bruikbare beschrijving van de cursusinhoud.

We hebben al gezegd dat het vaker gebruiken van ons raamwerk in andere situaties de invulling zou kunnen zijn van toekomstig werk dat voortborduurt op dit onderzoek. Dat is echter niet alles. Ook kan er nog een extra laag over deze specificatie en daarmee dit onderzoek heen worden gelegd. We zouden namelijk de strategieën kunnen implementeren in een computersysteem, waardoor je een soort decision support system krijgt. Hoewel dit niet binnen de scope van dit onderzoek valt, hebben we de mogelijkheid hiervan wel degelijk open gehouden. We hebben namelijk in dit onderzoek geprobeerd uitsluitend met technieken te werken die een sterke formele basis hebben. De modelleertalen 'Object Role Modelling' (ORM) en 'Yet Another Workflow Language' (YAWL) hebben deze formele basis. Een afstudeeronderzoek dat gelijktijdig met het onze uitgevoerd werd, toont aan dat een invulling als deze mogelijk is.

Een andere richting die toekomstig werk in zou kunnen gaan, heeft te maken met games. Dit is iets waar onze begeleider zich mee bezig zou willen houden en waar hij ons vast persoonlijk van op de hoogte heeft gesteld (Hoppenbrouwers, 2007). Het idee hiervan is om bijvoorbeeld de cursus of het vakgebied van de requirements engineering op te vatten als een spel. Onze specificatie zou hier dan de basis van kunnen zijn als specificatie van een spel bestaande uit subspelen of ‘opdrachten’. In de specificatie staat namelijk precies wat volgens ons de juiste stappen zijn om een requirements analyse in elkaar te zetten. Zo zouden we kunnen kijken hoe requirements engineers zelf te werk gaan en dit kunnen toetsen aan onze specificatie. Hierbij is ook een zeker scoringssysteem van belang. Hiermee zouden we requirements engineers met punten kunnen belonen als ze bepaalde stappen uit onze specificatie goed uitgevoerd hebben. Ook zouden we ze minpunten kunnen geven wanneer ze bijvoorbeeld een scenario opstellen waarin typeconcepten gebruikt zijn. Zo maak je het hele proces erg goed meetbaar en kun je bijvoorbeeld gaan kijken naar usability en speelbaarheid. Zeker een interessante volgende stap na dit onderzoek.

6. Lijst van afkortingen

Hoewel we zelf in dit document zoveel mogelijk proberen om geen afkortingen te gebruiken, gebeurt dit in de ICT wel veelvuldig. Zowel in officiële artikelen als in wat ruwere literatuur kom je afkortingen tegen. Om het overzicht te behouden, presenteren we in tabel 15 enkele veelgebruikte afkortingen.

Afkorting	Betekenis
BCoE	Basic Course of Events
BR	Business Rules
BRC	Business Rules Catalog
ConQuer	Conceptual Query
DB	Database
DM	Domain Model
DMP	Domain Model Population
FNWI	Faculteit der Natuurwetenschappen, Wiskunde en Informatica
GDM	Generic Domain Model
GUCD	Generic Use Case Diagram
IRIS	Information Retrieval and Information Systems
LISA-D	Language for Information Structure and Access Description
NIII	Nijmeegs Instituut voor Informatica en Informatiekunde
NORMA	Neumont ORM Architect
ORC	Object Role Calculus
ORM	Object Role Modeling
PS	Problem Statement
RA	Requirements Analysis
RE	Requirements Engineering
RIDL	Reference and Idea Language
SC	Scenario
TD	Terminological Definition
UC	Use Case
UCS	Use Case Survey
UCD	Use Case Diagram
UML	Unified Modeling Language
YAWL	Yet Another Workflow Language

Tabel 15: lijst van afkortingen

7. Bijlagen

Hier zijn twee bijlagen te vinden, behorende bij de online applicatie zoals beschreven in sectie 4.2.

7.1. Databasestructuur van de online applicatie

In deze sectie beschrijven we de drie tabellen waaruit de databasestructuur – die grotendeels gebaseerd is op het metamodel uit sectie 3.2.7 – van de applicatie bestaat. Van elke tabel beschrijven we de bijbehorende velden, het type daarvan en het doel daarvan onder ‘opmerkingen’.

7.1.1. Tabel ‘activity’

In deze tabel beschrijven we de activiteiten uit de specificatie.

Veld	Type	Opmerkingen
id	int(11)	Een uniek identificatienummer
activity_name	varchar(100)	De naam van de activiteit
parent_activity	int(10)	ID van de samengestelde stap boven deze activiteit
strategy_context	varchar(255)	Bestandsnaam van afbeelding van de strategy context
conceptual_description	varchar(255)	Bestandsnaam van afbeelding van de conceptuele beschrijving

Tabel 16: tabel ‘activity’

7.1.2. Tabel ‘activity_strategy’

In deze tabel beschrijven we de strategie van de activiteiten uit de specificatie.

Veld	Type	Opmerkingen
step_id	int(10)	Een uniek identificatienummer
activity_id	int(10)	ID van de activiteit waar de stap bij hoort
type	enum('ad-hoc', 'guided', 'composed')	Het type van deze stap
step_description	varchar(255)	De beschrijving van de stap zelf
listnumber	int(10)	Het nummer van de volgorde van de stap
is_immediate	enum('yes', 'no')	Wel of geen onmiddellijke volgorde
has_followup	enum('yes', 'no')	Wel of geen samengestelde stap
child_activity	int(10)	ID van de activiteit waar een samengestelde stap naar leidt

Tabel 17: tabel ‘activity_strategy’

7.1.3. Tabel ‘activity_notes’

In deze tabel beschrijven we de notes behorende bij *guided steps* uit de specificatie.

Veld	Type	Opmerkingen
id	int(10)	Een uniek identificatienummer
step_id	int(10)	ID van de stap waar deze note bij hoort
note_description	varchar(255)	De beschrijving van de note zelf

Tabel 18: tabel ‘activity_notes’

7.2. Programmatuur van de online applicatie

In deze sectie is in tabel 19 de programmeercode te vinden waarmee de activiteiten uit de database opgebouwd worden. Hoewel de online applicatie natuurlijk ook veel opmaak heeft in de vorm van html-bestanden, kiezen we ervoor enkel deze php-code weer te geven. Het geeft een goed beeld van hoe de databasestructuur er toe leidt dat de activiteiten juist op het web terechtkomen.

```
<?php
    include('connect.inc.php');
    $inputid = 1;

    if(isset($_GET['id']))
    {
        $inputid = $_GET['id'];
    }

    $query1 = "SELECT * FROM activity WHERE id = " . $inputid;
    $result1 = mysql_query($query1);

    $urlpictures = 'http://www.student.ru.nl/jeroenroelofs/MS/';
    $activityname = "";
    $strategycontext = "";
    $conceptualdescription = "";
    $activityid = "";
    $parentactivity = "";

    while($outcome1 = mysql_fetch_assoc($result1))
    {
        $activityname = $outcome1['activity_name'];
        $strategycontext = '';
        $conceptualdescription = '';
        $activityid = $outcome1['id'];
        $parentactivity = $outcome1['parent_activity'];
    }
    $location = '<a href="home.php?id=' . $activityid . '">' . $activityname . '</a>' . $location;

    do
    {
        $query1a = "SELECT * FROM activity WHERE id = " . $parentactivity;
        $result1a = mysql_query($query1a);

        while($outcome1a = mysql_fetch_assoc($result1a))
        {
            $location = '<a href="home.php?id=' . $outcome1a['id'] . '">' . $outcome1a['activity_name'] . '</a> &arr;' . $location;
            $parentactivity = $outcome1a['parent_activity'];
        }
    } while($parentactivity != '0');

    $query2 = "SELECT * FROM activity_strategy WHERE activity_id = " . $activityid . " ORDER BY listnumber, step_id ";
    $result2 = mysql_query($query2);

    $strategy = "";
    $is_followup = false;

    while($outcome2 = mysql_fetch_assoc($result2))
    {
        $closing = '<br>';
        $strategy .= "";

        if($outcome2['type'] == 'guided')
        {
            $strategy .= '<u>';
        }

        else if($outcome2['type'] == 'composed')
        {
            $strategy .= '<b>';
        }

        if(!$is_followup)
        {
            if($outcome2['is_immediate'] == 'yes')
            {
                $strategy .= '*';
            }

            if($outcome2['listnumber'] == '0')
            {
                $strategy .= '-';
            }
        }
    }
}
```


8. Zakenregister

<i>activity name</i>	19	<i>raw material</i>	20
afkortingenlijst	21	requirements engineering	10
alethic	16	scenario's	14
business rules	15	<i>sources</i>	20
<i>conceptual description</i>	26	<i>splits</i>	21
<i>conditions</i>	21	stadia.....	13
deontic.....	16	facade	13
depth-first.....	25	filled	13
domeinmodel	14	focused.....	13
domeinmodel populatie	15	<i>states</i>	20
functiepunten.....	11	<i>step</i>	22
generiek domeinmodel	15	ad-hoc	25
<i>intermediate products</i>	20	composed	25
<i>iteraties</i>	26	guided	25
literatuurstudie	28	<i>strategy</i>	22
metamodel	28	<i>strategy context</i>	20
method engineering	8	systeemontwikkeling	10
<i>multiple task</i>	22	<i>transition</i>	20
operationalisatie	11	use case diagram	13
ORC.....	16	use case survey	13
prioritering	23	use cases	12
met <i>immediate order</i>	24	<i>void</i>	21
met <i>order</i>	23	waterfall-methode	26
zonder <i>order</i>	23	YAWL.....	20
<i>products</i>	20		

9. Literatuur

- Booch, G., Rumbaugh, J., & Jacobson, I. (2005). *Unified Modeling Language User Guide*. Boston: Addison-Wesley Professional.
- Halpin, T. (2001). *Information Modeling and Relational Databases Publishers*. San Francisco: Morgan Kaufmann.
- Hoppenbrouwers, S. (2007). Persoonlijke communicatie. *Method Modelling as Game Design*.
- Hoppenbrouwers, S., Proper, E., & Van der Weide, T. (2005). Fact Calculus: Using ORM and Lisa-D to Reason About Domain. In R. Meersman, Z. Tari, & P. Herrero (Red.), *On the Move to Meaningful Internet Systems : OTM Workshops*. 3762, pp. 720–729. Cyprus: Lecture Notes in Computer Science.
- Jones, C. (1996). *Applied Software Measurement: Assuring Productivity and Quality* (2nd ed.). New York: McGraw Hill.
- Kruchten, P. (2000). *From waterfall to iterative lifecycle - a tough transition for project managers*. Whitepaper, Rational Software Corporation.
- Kulak, D., & Guiney, E. (2000). *Use Cases: Requirements in Context* (2nd ed.). New York: ACM Press.
- Meersman, R. (1982). *The RIDL Conceptual Language*. International Centre for Information Analysis Services. Brussel: Control Data Belgium, Inc.
- Mirbel, I., & Ralyté, J. (2006). Situational method engineering: combining assembly-based and roadmap-driven approaches. *Requirements Engineering*, 11, 58-78.
- Nuseibeh, B., & Easterbrook, S. (2000). Requirements engineering: a roadmap. (A. Finkelstein, Red.) *The Future of Software Engineering*.
- Proper, E. (1994). *ConQuer - 92 - The revised report on the conceptual query language LISA-D*. University of Queensland, Asymetrix Research Laboratory. Queensland: Brisbane.
- Ter Hofstede, A., Proper, E., & Van der Weide, T. (1993). Formal definition of a conceptual language for the description and manipulation of information models. *Information Systems*, 18 (7), 489–523.
- Van Bommel, P., Hoppenbrouwers, S., Proper, E., & Van der Weide, T. (2007). A Rule-Based Approach to Method Modelling. *Unpublished manuscript*.
- Van de Weerd, I., Brinkkemper, S., Souer, J., & Versendaal, J. (2006). A situational implementation method for web-based content management system-applications: Method engineering and validation in practice. *Special issue of Software Process: Improvement and Practice*. IEEE.
- Van der Aalst, W., & Ter Hofstede, A. (2005). YAWL: Yet Another Workflow Language. *Information Systems*, 30 (4), 245-275.