Radboud Universiteit Nijmegen

Het toepassen van

requirements management

op enterprise-architectuur

Master Thesis Informatiekunde / MT IK 126 / Chiel Schutter /

S0652318 / mail@chiel.net / Nijmegen, juli 2010

Colofon

Auteur:	Chiel Schutter
Studentnr.:	S0652318
Afstudeernr.:	MT IK 126
Contact:	mail@chiel.net
Titel:	Het toepassen van requirements management op enterprise-architectuur
Versie:	1.1 definitief
Plaats:	Nijmegen
Datum:	29/07/2010
Opleiding:	Master Informatiekunde
Instituut:	Nijmeegs Instituut voor Informatica en Informatiekunde (NIII)
Faculteit:	Faculteit der Natuurwetenschappen, Wiskunde & Informatica (FNWI)
Universiteit	Radboud Universiteit Nijmegen
Begeleider	dr. P. van Bommel (pvb@cs.ru.nl)
Tweede lezer	dr. S.J.B.A. Hoppenbrouwers (stijnh@cs.ru.nl)

Voorwoord

Ergens in april 1993 zei mijn vader plotseling: “Chiel, ik ga morgen een computer van mijn collega overkopen, een 386 met maar liefst 16mb werkgeheugen en 250mb hardeschijfruimte”. Ik dacht: “Geweldig, een computer, dat is interessant!”. Voetballen op het veldje werd plotseling ingeruild voor het leren van dos commando's en ik was niet meer weg te slaan van deze computer. Toen mijn vader twee jaar later zei: “Ik heb een internetabonnement afgesloten met World Online, dat is een soort bibliotheek waar je allemaal informatie kan opvragen” was het hek helemaal van de dam. Dit resulteerde in hoge internetkosten en een moeder die zich zorgen maakte over het feit of ik nog wel genoeg in de buitenlucht kwam. Het was voor mij meteen duidelijk en ik heb geen moment getwijfeld over de keuze van mijn studie.

Na het afronden van de MAVO heb ik gekozen voor de opleiding MBO Automatisering die ik vrij eenvoudig heb afgerond. Het knutselen met de hardware en het configureren van software vond ik interessant maar ik had sterk de behoefte om mij verder te specialiseren. De keuze voor de vervolgstudie Bachelor Informatica was dan ook een logisch gevolg. Tijdens deze studie is mij duidelijk geworden dat mijn passie niet zo zeer bij het programmeren en de technische aspecten lag maar dat ik meer geïnteresseerd was in het snijvlak tussen de bedrijfsprocessen en de techniek. Aangezien ik in 2007 bij het afronden van deze opleiding nog niet de behoefte had om te werken en het gevoel had dat ik nog een stapje hoger kon, ben ik de opleiding Informatiekunde gaan volgen. Ik ben zeer blij dat ik deze keuze heb gemaakt en ook trots dat ik deze opleiding succesvol heb afgerond. Hier ligt mijn Master Thesis voor u, waar ik veel tijd en energie heb ingestoken. Maar uiteraard was ik nooit zover gekomen zonder de hulp van anderen.

Ten eerste wil ik mijn begeleider dr. Patrick van Bommel in het bijzonder bedanken aangezien hij mij op de juiste momenten heeft weten te motiveren. De gesprekken heb ik altijd als zeer prettig ervaren en hebben mij geholpen om tot dit eindstation te komen. Dat deze gesprekken altijd werden gecombineerd met de benodigde dosis humor heb ik altijd enorm kunnen waarderen.

Ik wil drs. Daan Sauren bedanken voor alle tijd en energie die hij in mijn Master Thesis heeft gestoken. De kritische noten, hevige discussies en gesprekken hebben mij vaak aan het denken gezet en wat zich uiteindelijk heeft uitbetaald in een nog beter document.

Ik wil Freek van Workum, Joep Munsterman, Ian Piepenbrock, Koen Seegers, Charl van de Horst, Stephan van Hoorn, Daan Wijlens, Lenny Kamst, Michel Bluysen, Ronald Hock, Broem Bruggink, Patrick Lindeman, Jochem Sipkens en Frans Follings bedanken aangezien zij indirect aan mijn scriptie hebben bijgedragen. De momenten dat ik niet met mijn onderzoek bezig was en de behoefte had aan gezelligheid en een lekker pilsje, stonden zij altijd voor mij klaar. Ik bedank Sabine Brusse, waarmee ik heel wat dagen samen in het studielandschap heb doorgebracht. We hebben elkaar altijd goed weten te motiveren. Ik wil ook mijn vriendin Mariëlle Fiene bedanken voor haar steun en advies.

Tot slot bedank ik mijn familie. Met name mijn ouders die mij altijd hebben gesteund tijdens mijn studiecarrière. Ze hebben altijd achter mij gestaan, zonder hen was ik zeker niet zo ver gekomen!

Samenvatting

In deze Master Thesis heb ik onderzoek gedaan naar de mogelijkheid om requirements management op enterprise-architectuur toe te passen. Bij het bedrijven van enterprise-architectuur worden er modellen ontwikkeld welke inzicht geven in de technologiearchitectuur, applicatiearchitectuur en de bedrijfsarchitectuur. Deze modellen worden gevisualiseerd met de modelleertaal ArchiMate. Voor deze modellen worden eisen en wensen (requirements) geformuleerd. Door deze requirements te kunnen herleiden naar een dergelijk model en vice versa, kan de compliance worden getoetst en de consistentie van een architectuur gewaarborgd worden.

Het oorspronkelijke plan aan de start van dit onderzoek was, om de resultaten van mijn literatuurstudie te gebruiken als aanzet voor een requirementstaal voor enterprise-architectuur. Tijdens het schrijven van deze Master Thesis is er echter een onderzoek [QUAR091] [QUAR092] gepubliceerd waarin een requirementstaal voor enterprise-architectuur wordt gepresenteerd. Deze taal heet ARMOR en is ontwikkeld door Novay, BiZZdesign en de Universiteit van Twente en is een toevoeging voor ArchiMate. Deze taal heeft ervoor gezorgd dat een groot deel van mijn probleemstelling werd opgelost. De ontdekking van deze taal heeft mij ertoe gedwongen om de opzet en structuur van dit onderzoek enigszins aan te passen. De resultaten van mijn literatuurstudie heb ik gebruikt om deze taal te onderzoeken en een verbetering of uitbreiding aan te dragen.

Op basis van mijn literatuuronderzoek heb ik geconcludeerd dat de koppeling tussen enterprise-architectuur en het operationele niveau van een enterprise relevant is aangezien enterprise-architectuur beschrijft hoe de organisatie, de informatievoorziening, de applicaties en de infrastructuur hun vorm hebben gekregen en hoe zij zich voordoen in het gebruik. Ik heb een uitbreiding voor ARMOR ontwikkeld waarbij mogelijk wordt gemaakt om inzicht te verkrijgen in de functionaliteit van een systeem, applicatie of applicatiecomponent. Concreet heb ik een aantal relaties aan ARMOR toegevoegd waardoor het mogelijk is om een UML klassendiagram te koppelen met concepten van ArchiMate en ARMOR. Hierdoor kan men requirements op architectuurniveau die bepaald worden door principes en belangen van stakeholders, herleiden naar concrete functionele requirements op systeemniveau.

Inhoudsopgave

Lijst van Figuren

Lijst van Tabellen

1 Inleiding

1.1 Probleemstelling

Enterprise-architectuur wordt vaak gezien als een management-stuurinstrument om complexe veranderingen beter te beheersen, te besturen, te faciliteren en te controleren. Het bedrijven van enterprise-architectuur kan voor elke organisatie een ander zakelijk doel hebben. Bijvoorbeeld voor de ondersteuning van strategische keuzes, het besturen van veranderingen, het opsporen van een knelpunt, het vereenvoudigen en stroomlijnen van bedrijfsprocessen of voor het afstemmen van IT en bedrijfsvoering. Logisch gezien bestaat enterprise-architectuur uit een samenhangend geheel van concepten, principes, onderbouwingen en modellen die als leidraad worden gebruikt om de zojuist genoemde doelstellingen te kunnen bereiken binnen het gestelde normenkader. Deze enterprise-architectuur blauwdrukken worden ontwikkeld op basis van vooraf vastgestelde requirements. Deze requirements zijn gebaseerd op wensen, behoeften en eisen die door de organisatie zelf of door experts zijn samengesteld. Wanneer een model is ontwikkeld en de definitieve fase is bereikt, wil men controleren of dit model ook daadwerkelijk voldoet aan de vooraf vastgestelde requirements.

In deze Master Thesis ga ik proberen om een instrument te ontwikkelen met betrekking tot het toetsen van de compliance tussen requirements en een enterprise-architectuur model.

Dit resulteert in de volgende onderzoeksvraag:

Op welke manier is het mogelijk om de compliance tussen een enterprise-architectuur model en de bijbehorende requirements te toetsen?

Deze onderzoeksvraag verdeel ik onder in een aantal deelvragen:

- **1:** Hoe is enterprise-architectuur gepositioneerd binnen een organisatie?
- **2:** Op welke manier wordt een enterprise-architectuur beïnvloed en wordt deze ontworpen?
- **3:** Hoe wordt er gecommuniceerd op enterprise-architectuur niveau?
- **4:** Op welke manier is het mogelijk om requirements te herleiden?
- **5:** Aan welke eisen moet een requirementstaal voor enterprise-architectuur voldoen?
- **6:** Voldoet ARMOR, de requirementstaal voor enterprise-architectuur aan de gestelde eisen?

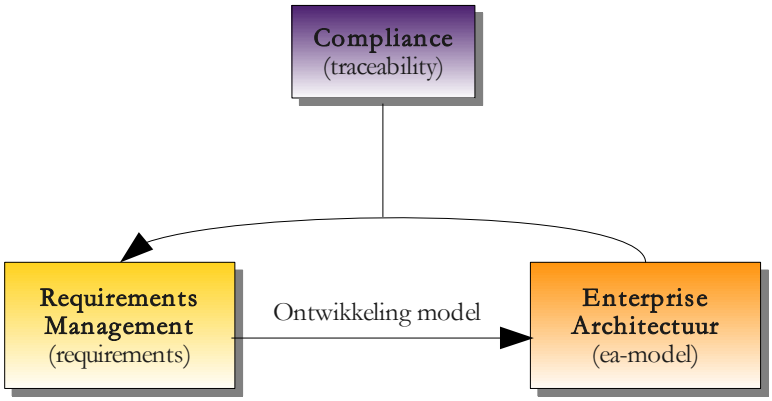
1.2 Verantwoording

Er gaat een hoop geld om in de IT-sector; een sector waarin niet alle uitgaven altijd goed terecht komen of het gewenste resultaat hebben. Hoe groter en complexer de projecten en de beslissingen die hierin genomen dienen te worden, hoe groter de kans is dat het eindresultaat verschilt van de voorafgestelde doelstellingen en eisen. Gezien het complexe ontwikkeltraject is het erg moeilijk, voornamelijk binnen enterprise-architectuur, ervoor te zorgen dat alle requirements terugkomen in het uiteindelijke product. Voor een project dat miljoenen euro's heeft gekost, is het voor de klant of het bedrijf zelf erg gewenst om er zeker van te zijn dat de vooraf vastgestelde eisen ook werkelijk terug te vinden zijn in het uiteindelijke enterprise-architectuur model. Dit model is de basis van een grote omslag binnen een bedrijf en zal dan ook tot in detail moeten kloppen en overeenkomen met de requirements.

Dit is van cruciaal belang om ervoor te zorgen dat het enterprise-architectuurmodel aan alle wensen voldoet en er geen geld over de balk gegooid zal worden door verder te gaan met een model dat niet voldoet aan de verwachtingen. Ook is de compliance van belang om te zorgen dat de bedrijfsvoering vanwege 'fouten' in het enterprise-architectuurmodel niet in het gedrang komt. Ook op dit gebied is een goed controletraject relevant.

1.3 Theoretisch kader

In grote lijnen valt te zeggen dat de probleemstelling is verdeeld over twee hoofdkennisgebieden. Aan de voorkant hebben wij te maken met de requirements en de totstandkoming en vorm hiervan, aan de achterkant begeven wij ons in het gebied van enterprise-architectuur en modellering. Tussen de voor- en achterkant zal ik trachten een brug te bouwen waarmee de compliance getoetst kan worden.



Figuur 1: Model kennisgebieden

1.3.1 Enterprise-architectuur

Alhoewel dit kennisgebied uitgebreid behandeld wordt en de lezer de nodige achtergrondinformatie wordt aangeboden, zal de focus van dit onderzoek liggen op het enterprise-architectuurmodel en niet op het ontwikkelingsproces hiervan. Een enterprise-architectuurmodel wordt ontwikkeld met een modelleertaal. In dit onderzoek hanteer ik de aanname dat dit model met de modelleertaal ArchiMate is ontwikkeld. ArchiMate levert een geünificeerde modelleertechniek op het gebied van enterprise-architectuur en is onlangs geaccepteerd als standaard door de standaardbeheerorganisatie 'The Open Group'. Om de scope van het onderzoek te beperken, zal ik mij voornamelijk richten op modellen die betrekking hebben op de applicatielaag van een enterprise. In deze Master Thesis hanteer ik de volgende definitie wanneer ik praat over een enterprise-architectuur model:

Ea-model is een enterprise-architectuurmodel van de applicatielaag en ontwikkeld in de modelleertaal ArchiMate. Dit model geeft inzicht in de architectuur van het applicatielandschap van een enterprise.

1.3.2 Requirements Management

De requirements bevinden zich in het kennisgebied requirements management. In deze Master Thesis worden verschillende technieken en talen op gebied van requirements management behandeld. Vooralsnog wordt er geen aanname gehanteerd hoe deze requirements zijn geformuleerd.

Requirements zijn eisen en wensen van het ea-model die zijn samengesteld vooraf aan het ontwikkelingsproces van het ea-model.

Requirementstaal is de taal waarin de requirements zijn geformuleerd.

1.3.3 Compliance

Het compliance kennisgebied is het linken van de achterkant van het onderzoeksdomein met de requirements aan de voorkant. De Engelse definitie van compliance is volgens de Van Dale “volgzaamheid, inschikkelijkheid, meegaandheid, toegeeflijkheid en gehoorzaamheid” [VAND09]. Vooral de term gehoorzaamheid kent een juridische kant, namelijk de bekende wetgeving die in 1990 in Nederland is ingevoerd in de financiële sector. Enterprises krijgen regels opgelegd van zowel de overheid als de financiële toezichthouders. In Nederland zijn dit de DNB (De Nederlandsche Bank),

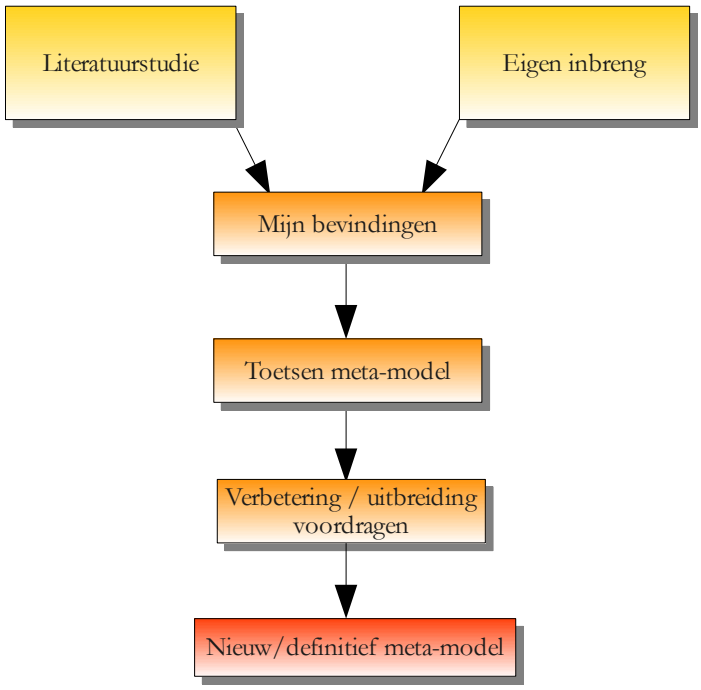
AFM (Autoriteit Financiële Markten) en de PVK (Pensioen en Verzekeringkamer). Enterprises bevorderen en waarborgen door middel van compliance de integriteit. Van Dale geeft dan ook de volgende Nederlandse definitie: "Het naleven van gedragsregels binnen een organisatie, bijvoorbeeld met betrekking tot de effectenhandel" [VAND09]. De wetgeving zorgt onder andere voor het bestrijden van normafwijkend gedrag, fraude en witwassen.

In dit onderzoek heeft de term compliance een ietwat afwijkende betekenis. Er zijn overeenkomsten, namelijk de integriteit, overeenstemming, naleving en het uitvoeren van een controle. In deze Master Thesis is er geen sprake van een wetgeving, maar de voorafgestelde requirements die bij een ea-model horen. In deze Master Thesis wordt de volgende definitie gehanteerd:

Compliance is het linken, herleiden en controleren van de overeenkomstigheid tussen een ea-model en de requirements zodat de integriteit van de architectuur wordt gewaarborgd en bevorderd.

1.4 Methode

In dit onderzoek ben ik van plan om de hoofdvraag te gaan beantwoorden met behulp van een aantal deelvragen. In het beantwoorden van deze vragen laat ik me primair leiden door literatuur die aansluit bij het onderwerp. Na elk hoofdstuk zal ik mijn bevindingen behandelen. De literatuur gecombineerd met mijn eigen inbreng vormen de basis voor het toetsen van het meta-model. Het meta-model zal ik toetsen door middel van een aantal praktische voorbeelden. Bij ieder van deze methoden zal ik erop toezien dat de validiteit en betrouwbaarheid bewaakt blijft. Het onderstaande figuur illustreert mijn methode door middel van een onderzoeksmodel.



Figuur 2: Onderzoeksmodel

2 Enterprise-architectuur

In dit hoofdstuk wordt de definitie van enterprise-architectuur behandeld. Enterprise-architectuur is een belangrijk kennisgebied binnen dit onderzoek. Het is belangrijk om te weten waaruit enterprise-architectuur bestaat en welke elementen relevant zijn voor dit onderzoek. Zoeken met Google naar de term "enterprise architecture" levert maar liefst 7.470.000 resultaten op. Wat houdt het precies in en welke definitie wordt in deze Master Thesis gehanteerd?

2.1 Definitie

2.1.1 Enterprise

Bedrijven, organisaties, ondernemingen, gouvernementele instellingen zijn allemaal enterprises. [HOOG07, p. 33] In de literatuur zijn vele definities van de term 'enterprise' te vinden, waarbij altijd een aantal specifieke kenmerken terug te vinden zijn. In deze Master Thesis gebruik ik de definitie zoals deze door Daft is gehanteerd:

> "**Organizations** are social entities that are goal directed, are designed as deliberately structured and coordinated activity systems, and are linked to the external environment." [DAFT01, p. 12]

In deze definitie komen vier specifieke kenmerken naar voren [HOOG07, p. 34]:

- Een enterprise is een sociale entiteit, aangezien mensen een belangrijke rol spelen ondanks de technologische hulpmiddelen. Mensen blijven namelijk verantwoordelijk voor het realiseren van het doel of de doelen.
- Een enterprise bezit doelgerichtheid. Wanneer een sociale entiteit doelgericht te werk gaat, dan ontstaan er bepaalde interactiepatronen, waarbij het gaat om coördinatie, coöperatie en collaboratie gezien in het kader van de enterprisedoelstelling.
- Omdat deze interactiepatronen niet vanzelf ontstaan kunnen we concluderen dat een enterprise bewust is en met een zekere bedoeling geconstrueerd is.
- Het laatste kenmerk is dat een enterprise een grens heeft (de enterprisegrens) die ervoor zorgt dat deze te onderscheiden valt van haar omgeving. Dit onderscheid is relevant omdat een enterprise enerzijds zijn functie levert aan (entiteiten in) de omgeving en anderzijds energie, resources en informatie uit de omgeving betrekt.

Enterprises bestaan al vanaf het begin van de mensheid. De militaire en gouvernementele bestuursfuncties binnen het Romeinse Rijk zijn te beschouwen als bekende, vroege organisatievormen. [HOOG07, p. 35]

2.1.2 Architectuur

In het verleden manifesteerde complexiteit zich allereerst op het gebied van de bouwkunde, het kennisgebied dat structuur en orde bracht bij het realiseren van bouwwerken [SOET04, p. 8]. Vitruvius (bouwmeester van Julius Caesar) schreef tien boeken over architectuur en gaf in de eerste eeuw voor Christus drie aspecten hieraan, namelijk: 'utilitas', 'firmitas' en 'venustas', beter bekend als het drieluik van Vitruvius. Utilitas staat voor de gebruiksaspecten: doelmatigheid, nuttigheid en deugdelijkheid. Firmitas staat voor fysieke zaken: duurzaamheid, vastheid, en sterkte. Venustas staat voor bekoorlijkheid en uiterlijk schoon, dus de beleving. Kortweg gesteld een 'gebouw', moet stevig van structuur zijn, moet de beoogde functie vervullen en voldoen aan de eisen van schoonheid [SOET04, p 8]. Hetzelfde geldt voor een organisatie, deze behoort dienstbaar te zijn aan haar doelstelling, dient solide te zijn en met kwaliteit in het functioneren [SOET04, p. 9].

Rijsenbrij vertaalt het drieluik van Vitruvius op de volgende wijze naar de digitale wereld [RIJS04, p. 8]:

- de functionaliteiten en hun onderlinge samenhang (de gebruiksmogelijkheden van het artefact);
- de gebruikte componenten, technologieën en de integratietechnieken (voor een degelijke constructie);
- het uiterlijke gedrag, de beleving door gebruikers (voor de sfeerbepaling, het gebruiksplezier).

De term architectuur bestaat al enige tijd binnen de IT-sector. Volgens velen (Rijsenbrij, Truijens) is het begrip architectuur in 1972 door Blaauw geïntroduceerd in de digitale wereld. Het ontwerpen van het OS/360 operating system bestond namelijk uit drie elementen; architectuur, implementatie en realisatie. Hier werd architectuur apart gedefinieerd voor één enkel computerontwerp, waarin de functionaliteit van een systeem werd beschreven voor de eindgebruiker. Het architectuurconcept werd door Jacques Theeuwes (Theeuwes, 1987) en Jan Truijens (Truijens et al., 1990) in de jaren tachtig toegepast op het niveau van informatievoorziening.

Dietz beschrijft de werkzaamheden van een architect als "Architectureren is het bedenken van de conceptuele kaders waarbinnen degenen die meer concreet bezig zijn met bedrijfsprocessen, applicaties of infrastructurele componenten, dienen te opereren. Architectureren is eerst en vooral het creëren van die architecturen, maar omvat ook het maken van plannen en het uitzetten van strategieën." [DIET98,

p. 6]

2.2 Stromingen

In de literatuur zijn twee verschillende stromingen te onderscheiden van enterprise-architectuur. Namelijk de descriptieve (beschrijvende) en prescriptieve (voorschrijvende) benadering.

2.2.1 Descriptief

De descriptieve benadering stelt dat architectuur een feitelijke beschrijving is van de huidige karakteristieken van artefacten en gaat al voornamelijk uit van de oplossingsmogelijkheden hierbinnen. [HOOG04, p. 3] De bestaande situatie wordt hierbij in kaart gebracht. Een descriptieve definitie van architectuur wordt gegeven door Zachman en heeft veel invloed gehad op de architectuurwereld:

"Architecture is that set of design artifacts, or descriptive representations, that are relevant for describing an object such that it can be produced to requirements (quality) as well as maintained over the period of its useful life (change)." [ZACH96, p. 5]

Booch, Rumbaugh, and Jacobson geven de volgende definitie:

"An architecture is the set of significant decisions about the organization of a software system, the selection of the structural elements and their interfaces by which the system is composed, together with their behavior as specified in the collaborations among those elements, the composition of these structural and behavioral elements into progressively larger subsystems, and the architectural style that guides this organization---these elements and their interfaces, their collaborations, and their composition." [BOOC99, p. 36-37]

Een meer recentere descriptieve definitie wordt gegeven door Bass, L., Clements, P., and Kazman:

"The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them." [BASS03, p. 3]

De IEEE (Institute of Electrical and Electronics Engineers) beschrijft het op de volgende wijze:

"The fundamental organization of a system embodied by its components, their relationships to each other and the environment, and the principles guiding its design and evolution". [MAIE01, p. 107]

Er zijn meerdere descriptieve definities van architectuur te vinden, maar deze zijn vrijwel identiek aan de zojuist gegeven vier definities. In de bovenstaande definities wordt architectuur beschouwd als een systeem. De descriptieve benadering is dus een benadering vanuit engineering, met het grote gevaar dat

het belevingsaspect onderbelicht wordt. [RIJS05, p. 3].

2.2.2 Prescriptief

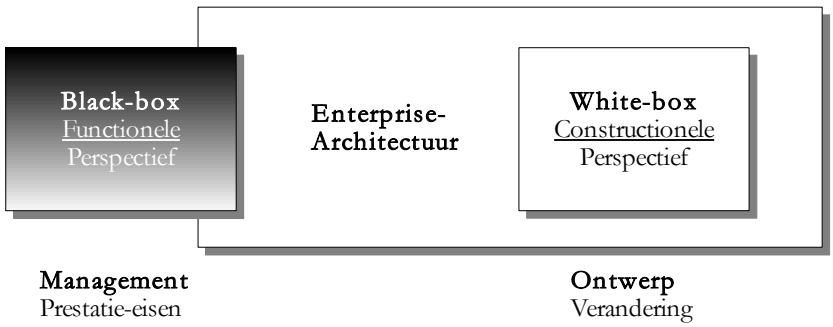
Volgens The Open Group bestaat enterprise architectuur uit twee definities [THEO091]:

- A formal description of a system or a detailed plan of the system at component level to guide its implementation.
- The structure of components, their inter-relationships and the principles and guidelines governing their design and evolution over time.

De eerste definitie van The Open Group is descriptief, de tweede definitie is echter niet descriptief. Architectuur wordt hier gedefinieerd als "de principes en richtlijnen die hun ontwerp en evolutie bepalen door de tijd heen". Dit concept wordt ook wel prescriptief genoemd.

Bij de prescriptieve benadering zijn een verzameling van principes en richtlijnen leidend voor toekomstige artefacten. Het essentiële verschil tussen de twee benaderingen ligt in het feit dat de prescriptieve benadering uitgaat van de vraagkant, terwijl de descriptieve benadering uitgaat van de mogelijke oplossing. [RIJS05, p. 3].

Hoogervorst heeft een prescriptieve visie op enterprise-architectuur en definieert dit als volgt: "Een consistente en coherente set van principes en standaarden die aangeeft hoe de enterprise moet worden ontworpen." [HOOG07, p 194] Hij benadrukt enterprise-architectuur als een belangrijke communicatieve brugfunctie, omdat principes en standaarden voor ontwerp (constructioneel perspectief) kunnen worden gekoppeld aan, beredeneerd uit en gerechtvaardigd door de eisen vanuit het functionele perspectief. Enterprise-architectuur wordt op deze manier gezien als een stuur- en communicatiemiddel om op een hoog abstractieniveau te kunnen redeneren over het ontwerp van de enterprise waarin een verzameling (set) van principes en richtlijnen leidend zijn voor het functionele en constructieve perspectief. Deze communicatieve brugfunctie kan schematisch weergegeven worden.



Figuur 3: Enterprise-architectuur als communicatiebrug [HOOG07]

Sogeti verstaat onder enterprise-architectuur het volgende:

"Een consistent geheel van principes en modellen dat richting geeft aan ontwerp en realisatie van de processen, organisatorische inrichting, informatievoorziening en technische infrastructuur van een organisatie." [WAGT01]

Rijsenbrij heeft de volgende opvatting van (digitale) architectuur:

"Een coherente, consistente verzameling principes, verbijzonderd naar uitgangspunten, regels, standaarden en richtlijnen, die beschrijft hoe een onderneming, de informatievoorziening, de applicaties en de infrastructuur hun vorm hebben gekregen en hoe zij zich voordoen in het gebruik." [RIJS04, p. 6]

De descriptieve benadering is belangrijk, maar refereert naar mijn mening direct naar het systeemontwerp. Het is veel interessanter om te kijken hoe iets gerealiseerd dient te worden. De prescriptieve benadering is bewust op zoek naar expliciete ontwerpprincipes (standaarden, richtlijnen, regels) die een consistent en coherent ontwerp mogelijk maken.

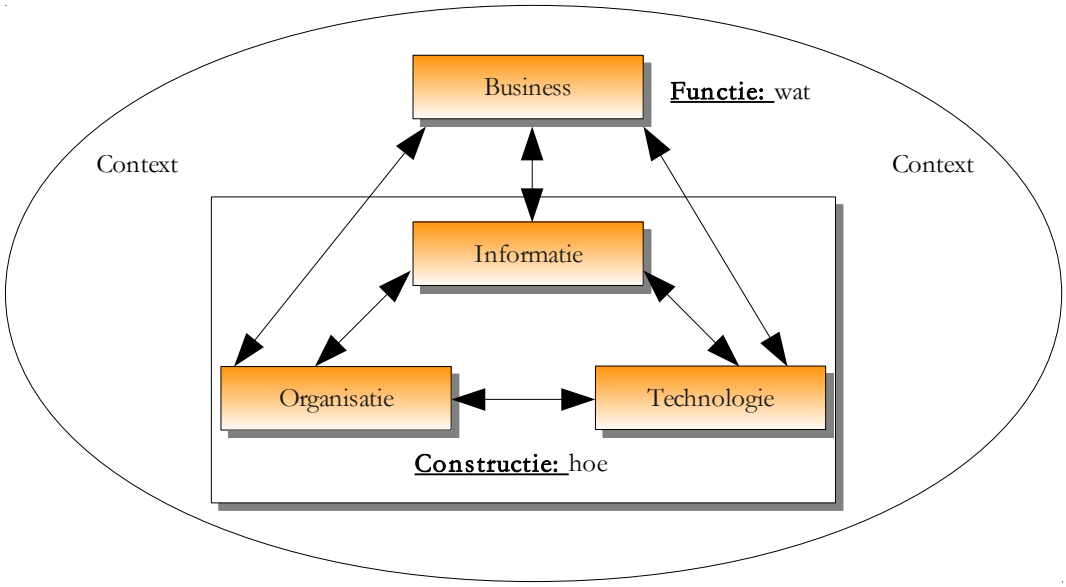
In deze Master Thesis gebruik ik de definitie die door Lankhorst wordt gehanteerd. Lankhorst houdt rekening met de verschillende deelarchitecturen en zowel de prescriptieve als de descriptieve benadering komt terug in deze definitie.

Enterprise architecture: a coherent whole of principles, methods, and models that are used in the design and realisation of an enterprise's organisational structure, business processes, information systems, and infrastructure. [LANK05, p. 3]

2.3 Domeinen

Zoals u in de bovenstaande definitie kan afleiden worden architecturen in verschillende domeinen gebruikt. Voorbeelden zijn de business-architectuur, applicatie-architectuur, infrastructuur-architectuur en informatie-architectuur. Bij enterprise-architectuur kijkt men niet naar één domein, maar wordt het beheersen van de gehele bedrijfsvoering nagestreefd. Elk ontwerp van de onderneming dan wel haar ondersteuning met IT-middelen, begint met een verzameling architectuurprincipes, die als het ware de ontwerpruimte inperkt [RIJS04, p. 6]. Principes zijn richtinggevende uitspraken ten behoeve van essentiële beslissingen en vinden vaak hun oorsprong in strategie en organisatiecultuur van de onderneming. Veel bestaande architectuur-initiatieven beperken zich tot het vastleggen van architectuurmodellen en principes per architectuurdomein [BERG04]. Een architectuur bestaat in dat geval uit losse onderdelen waarbij er geen relaties tussen andere architecturen zichtbaar zijn. Er is dan sprake van een geïsoleerde architectuur. Het is relevant dat de onderlinge relaties tussen de architecturen worden vastgelegd.

Hoogevorst maakt onderscheid in vier hoofdontwerpdomeinen waarbij elk domein een architectuur bevat. Enterprise-architectuur bestaat dus uit vier deelarchitecturen. [HOOG07, p. 198]:



Figuur 4: Enterprise deelarchitecturen [HOOG07, p. 198]

2.3.1 Business domein & architectuur

Het businessdomein is de primaire enterprisefunctie. Hier gaat het onder andere om de producten en diensten, klanten, het economische model en de relatie met de omgeving. De businessarchitectuur geeft

richting aan hoe het businessdomein moet worden geëxploiteerd en geëxploreerd. [HOOG07, p. 199]

2.3.2 Organisatie domein & architectuur

Het vaststellen van de primaire enterprisefunctie laat nog een bepaalde graad van vrijheid over: de vraag hoe de productie en de daarmee samenhangende activiteiten voor de levering van de producten en diensten feitelijk worden georganiseerd. Bij dit domein gaat het om de interne inrichting (constructie) van de enterprise, bijvoorbeeld processen, gedrag van medewerkers, organisatiecultuur, management, humanresourcesmangement en de verschillende structuren en systemen. De organisatiearchitectuur geeft normatieve sturing aan de organisatorische inrichting. [HOOG07, p. 199]

2.3.3 Informatie domein & architectuur

Binnen het business- en het organisatiedomein is informatie een cruciale factor voor alle aspecten die binnen deze domeinen een rol spelen. Bij informatie is de structuur en kwaliteit van informatie belangrijk, maar ook management van informatie en het gebruik van deze informatie. De informatiearchitectuur schrijft voor hoe informatie moet worden gehanteerd. [HOOG07, p. 199-200]

2.3.4 Technologie domein & architectuur

Technologie is essentieel voor de ondersteuning van bedrijfsprocessen, organisatie, informatie en ook voor toekomstige ontwikkelingen. Het technologiedomein is dus ook een belangrijk constructioneel ontwerpdomein. De met een specifieke technologie samenhangende architectuur heeft betrekking op de normatieve sturing van de met die technologie te ontwikkelen systemen. Voor informatietechnologie geldt dan de IT-architectuur, die voorschrijft hoe IT-systemen moeten worden ontworpen. [HOOG07, p. 200]

2.3.5 Ontwerpdomeinen & principes

Elk domein bevat meerdere ontwerpdomeinen. In tabel 1.1 wordt een overzicht gegeven van mogelijke IT-ontwerpdomeinen [HOOG07, p. 208]:

IT-ontwerpdomeinen		
Colaboratie Processen Services	**E-business** Kanalen/interfaces Processen Transacties	**Data Management** Meta data Validatie Modellen
Integratie Middleware Services Flow control	**Security** Authenticatie Autorisatie Perimeter defence	**Data Warehouse** Toegang Operatie (ETL) Inrichting
Netwerk Protocollen Toegang	**Platform** Opslag/beveiliging Configuratie Schaalbaarheid	**Applicaties** Structuur Communicatie Interactie

Tabel 1: IT-ontwerpdomeinen [HOOG07, p. 208]

Een overzicht van mogelijke principes binnen een IT-architectuur [HOOG07, p. 207]:

- E-business oplossingen moeten toegangskanaalonafhankelijk zijn.
- Operationele data moeten geschieden zijn van informatieve data.
- Integratieservices mogen geen businesslogica bevatten.
- Het gebruik van een IT-service moet onafhankelijk zijn van de service-implementatie.
- Toegang tot systemen moet gebaseerd zijn op authenticatie en rolgebaseerde autorisatie.
- Content en presentatie moeten gescheiden worden.
- Alle berichtdefinities moeten een gedocumenteerde inhoud hebben.

Zoals ik al eerder vermeld hebt en Hoogevorst tevens benadrukt, is de samenhang en integratie tussen de verschillende architecturen zeer belangrijk. Deze dienen ontworpen te worden door middel van een coherente set van principes en standaarden.

2.4 Raamwerken

Zoals uit de bovenstaande paragraaf geconcludeerd kan worden, bestaat een enterprise uit domeinen

met daarin meerdere deelarchitecturen. Deze deelarchitecturen worden ontworpen op basis van principes en de daaruit voortvloeiende uitgangspunten, regels, standaarden en richtlijnen. Het is een lastige klus voor een architect om al deze te kunnen beheersen. Hiervoor gebruiken architecten architectuurraamwerken of ontwikkelen een eigen raamwerk voor de organisatie. Naast architectuurprincipes kunnen natuurlijk ook metamodellen, ontwerpfragmenten en uiteindelijk de services of informatiesysteemfuncties op een dergelijke manier worden opgeborgen. [RIJS05, p. 11] Alle architecturen krijgen in een architectuurraamwerk een plaats en worden gecontroleerd op consistentie en coherentie. Bekende architectuurraamwerken zijn TOGAF (The Open Group) en IAF (Cap Gemini).

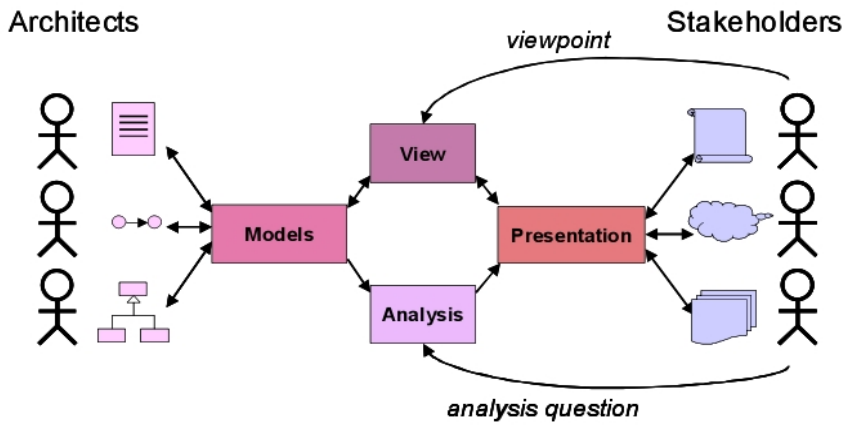
2.5 Communicatie

Communicatie is een belangrijk element, aangezien enterprise-architectuur draait om de juiste informatie verkrijgen, analyseren, verwerken en presenteren. Modellen, analyses, views en presentaties zorgen voor een goede communicatielijn tussen architect en de belanghebbende [LANK05, p. 4]. Een belanghebbende, ook wel stakeholder genoemd, speelt een belangrijke rol binnen enterprise-architectuur. Deze is niet zo zeer geïnteresseerd in de architectuur op zich maar in de impact die de architectuur kan hebben op zijn belangen. Een stakeholder heeft meestal zijn eigen zienswijze, ook wel viewpoint. Een architect moet zich de belangen van een stakeholder realiseren en deze met hem bespreken. Een architect moet altijd de architectuur kunnen verantwoorden aan de stakeholders binnen een enterprise.

Ik handhaaf in deze Master Thesis de definitie die Lankhorst hanteert:

> **Stakeholder:** een individu, team of organisatie (of klasses hiervan) met interesses of belangen in een systeem. [LANK05, p. 2]

De communicatie binnen enterprise-architectuur kan op de volgende wijze schematisch weergegeven worden:



Figuur 5: Communiceren over enterprise architectuur [LANK05, p. 4]

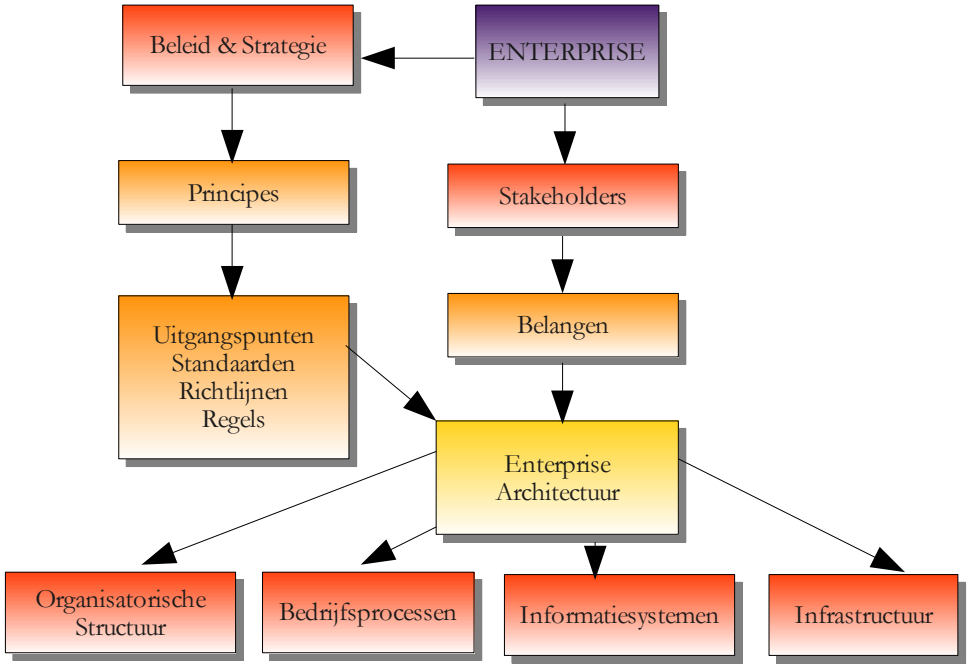
Om een geïntegreerd perspectief van een architectuur te ontwikkelen en deze op een coherente wijze te beschrijven en te presenteren aan de stakeholders, is er een techniek nodig. Deze techniek is ArchiMate, een taal voor enterprise-architectuurmodellering. In het volgende hoofstuk zal ik deze taal behandelen.

2.6 Mijn bevindingen

Op basis van dit hoofdstuk kan ik de volgende deelvragen beantwoorden:

- **deelvraag 1:** hoe is enterprise-architectuur gepositioneerd binnen een organisatie?;
- **deelvraag 2:** op welke manier wordt een enterprise-architectuur beïnvloed en wordt deze ontworpen?

Enterprise-architectuur speelt zich af op een hoog niveau binnen een enterprise. Architecturen worden ontwikkeld op basis van principes en belangen van stakeholders. De principes komen voort uit het beleid en de strategie welke verbijzonderd worden naar uitgangspunten, regels, standaarden en richtlijnen. Deze principes en belangen worden vervolgens gebruikt voor het ontwerp en de realisatie van de organisatorische structuur, bedrijfsprocessen, informatiesystemen en infrastructuur van een enterprise. Dit wordt in het onderstaande model weergegeven:



Figuur 6: Invloed op en van enterprise-architectuur

Enterprise-architectuur kan in twee vormen worden bedreven. Descriptief, welke een feitelijke beschrijving is van de huidige situatie en men voornamelijk uit gaat van de oplossingsmogelijkheden hierbinnen. En prescriptief, waarbij een verzameling van principes leidend zijn voor toekomstige artefacten en men uitgaat van de vraagkant. Binnen dit onderzoek is het niet van belang of het ea-model door middel van een prescriptieve of descriptieve benadering tot stand is gekomen.

De integratie, samenhang en communicatie tussen verschillende domeinen en architecturen staat bij enterprise-architectuur centraal. Requirements die opgesteld worden voor een dergelijk ea-model, zijn dus niet gebaseerd op een specifiek systeem, maar voor verschillende artefacten in verschillende lagen. Aangezien de strategie van een enterprise als leidraad wordt gebruikt voor het ontwerp van een architectuur, lijkt het erop dat de requirements zeer dichtbij ,of zelfs overeen komen, met de doelen en principes van een enterprise.

3 ArchiMate: modelleertaal voor EA

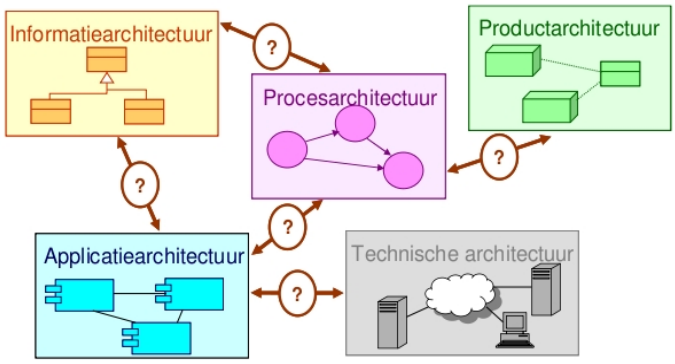
In dit hoofdstuk wordt uitgelegd wat het doel is van ArchiMate en waaruit het ArchiMate framework bestaat. Dit hoofdstuk is volledig gebaseerd op de documentatie en best practices van de ArchiMate Foundation [BOSM05] [BERG07] en het werk van Lankhorst [LANK05]. Dit hoofdstuk geeft duidelijkheid over het doel van de taal en welke concepten en mogelijkheden de taal biedt.

3.1 Doel van ArchiMate

Communicatie is zeer belangrijk bij het bedrijven van enterprise-architectuur. In het vorige hoofdstuk is toegelicht dat een stakeholder en een architect modellen, viewpoints en analyses gebruiken om over architectuur te kunnen communiceren. ArchiMate biedt hiervoor een gereedschapskist. Het ArchiMate-project is in 2001 gestart met als aanleiding om architecten te ondersteunen in hun ontwerpwerkzaamheden. In 2008 is ArchiMate officieel overdragen aan de standaardbeheerorganisatie The Open Group.

3.2 ArchiMate Framework

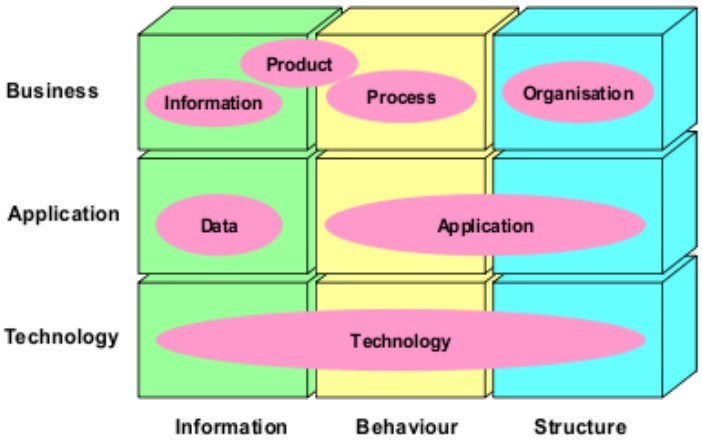
Zoals ik in het vorige hoofdstuk heb uitgelegd, bestaat een architectuur uit verschillende deelarchitecturen en domeinen en zijn de onderlinge relaties zeer belangrijk. ArchiMate beschrijft dit als inter-domein-relaties en geeft de mogelijkheid om de globale structuur binnen een domein en de relaties tussen domeinen te modelleren.



Figuur 7: Inter-domein-relaties ArchiMate [BOSM05]

Daarnaast is het mogelijk om modellen op verschillende manieren te visualiseren, gericht op verschillende belanghebbenden (verschillende viewpoints van stakeholders). Het ArchiMate framework bestaat uit twee dimensies, namelijk de verschillende lagen (business, applicatie en technologie) en de de

aspecten (informatie, gedrag, structuur). Hieronder wordt het modelleer-framework van ArchiMate weergegeven:



Figuur 8: ArchiMate modelleer-framework [QUAR092, p.26]

3.3 De lagen

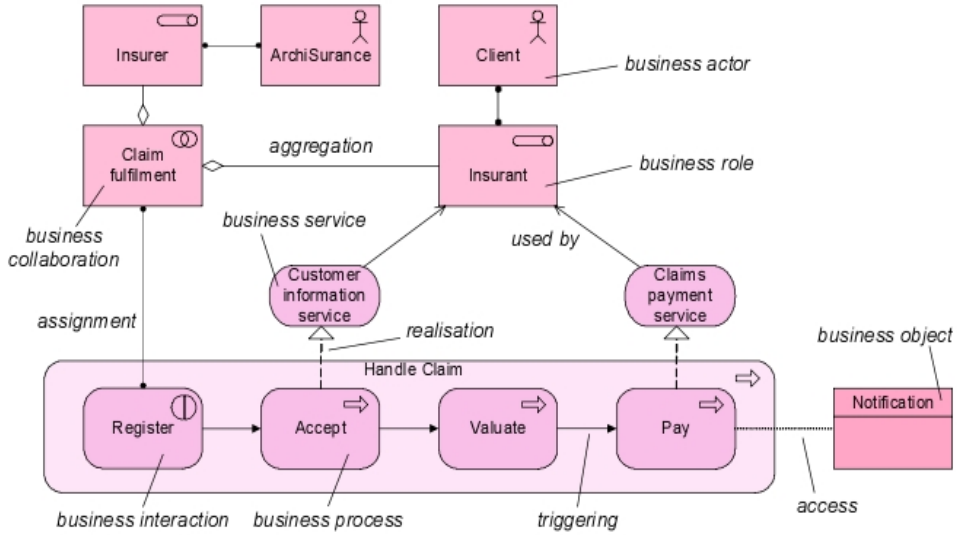
De verticale dimensie word onderverdeeld in drie verschillende lagen: business (bedrijfsprocessen), applicatie (software) en technologie (it-infrastructuur). Op elk van die lagen is een aantal concepten en relaties gedefinieerd. Totaal zijn er ongeveer 31 concepten en 12 relaties waarbij er is gekozen voor een service-georiënteerde insteek.

3.3.1 De bedrijfslaag

Deze laag biedt producten en services aan externe klanten. Deze services worden intern gerealiseerd door bedrijfsprocessen die uitgevoerd worden door bedrijfsactoren.

Categorie	Concepten van de bedrijfslaag
Structuur	business actor, business role, business collaboration, business interface, business object, business representation.
Gedrag	business service, business activity, business process, business function, business interaction, business event.
Hoog niveau	product, contract, value, meaning

Tabel 2: Concepten van de bedrijfslaag



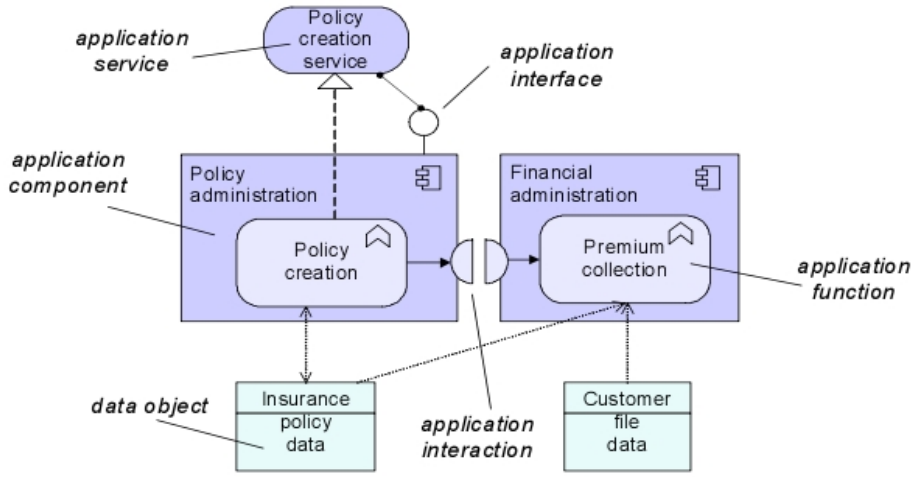
Figuur 9: Voorbeeldmodel bedrijfslaag [BOSM05]

3.3.2 De applicatielaag

Deze laag ondersteunt de bedrijfslaag met applicatieservices die worden gerealiseerd door (software) applicaties.

Categorie	Concepten van de applicatielaag
Structuur	Application component, application collaboration, application interface, data object.
Gedrag	Application service, application function, application interaction.

Tabel 3: Concepten applicatielaag



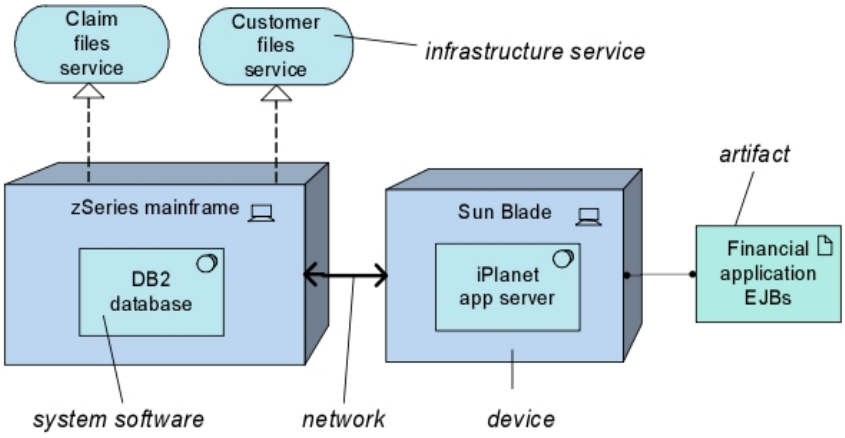
Figuur 10: Voorbeeldmodel applicatielaag [BOSM05]

3.3.3 De technologielaag

Deze laag biedt infrastructurele services die nodig zijn om applicaties te executeren en worden gerealiseerd door computer- en communicatie-hardware en systeemsoftware.

Categorie	Concepten van de Technologielaag
Structuur	Node, device, system software, artifact, infrastructure interface, communication path, network
Gedrag	Infrastructure service

Tabel 4: Concepten technologielaag



Figuur 11: Voorbeeld technologielaag model [BOSM05]

Het is eventueel nog mogelijk om een afzonderlijke laag boven de businesslaag te plaatsen die gericht is op externe klanten die gebruik maken van organisatorische services (echter kan dit ook als onderdeel van de businesslaag worden gezien). Binnen een laag kunnen weer deellagen onderkend worden (vertaald naar de visie van Hoogevorst: deelarchitecturen).

3.4 De aspecten

De horizontale dimensie bestaat uit drie verschillende aspecten:

- Structuur: hiertoe behoren de actoren (systemen, componenten, mensen, afdelingen etc.).
- Gedrag: dit aspect bestaat uit het gedrag (processen, services) wat door de actoren wordt uitgevoerd.
- Informatie: dit aspect bevat de kennis van het probleemdomein. Deze kennis wordt gebruikt door de actoren om te kunnen communiceren op basis van hun gedrag.

De structurering van dimensies bied de mogelijkheid om een enterprise vanuit meerdere gezichtspunten (viewpoints) te modelleren.

3.5 Service georiënteerd

Het concept "service" wordt gedefinieerd als een eenheid van functionaliteit die een bepaalde actor beschikbaar stelt aan zijn omgeving. Services zijn verschillend van aard en kunnen door zowel organisatorische eenheden worden aangeboden als technische faciliteiten. Service-oriëntatie is op een natuurlijke wijze verbonden met een opdeling in lagen waarbij hogere lagen gebruik maken van services die beschikbaar worden gesteld door lagere lagen. ArchiMate sluit op deze manier aan op de trend van Service Oriented Architecture.

3.6 Relaties

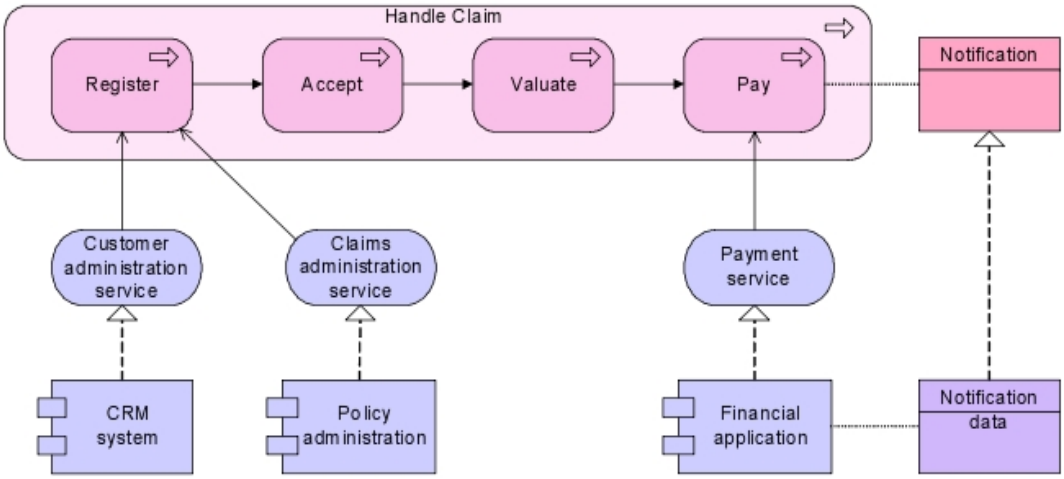
In elk van de zojuist behandelde lagen, worden verschillende relaties gebruikt om concepten met elkaar te kunnen koppelen. Deze relaties kunnen worden onderverdeeld in (1) structurele relaties die de structurele samenhang tussen concepten in beeld brengen, (2) dynamische relaties die worden gebruikt om de (temporele) afhankelijkheden tussen gedragsconcepten in beeld te brengen.

Categorie	Relaties
Structure relaties	Association, access, used by, specialisation, realisation, assignment, aggregation, composition, grouping
Behaviour relaties	Triggering, Flow, Junction.

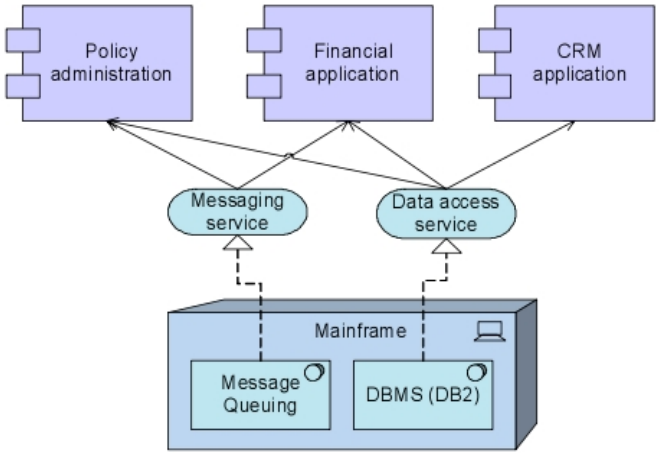
Tabel 5: ArchiMate relaties

3.7 Afstemming van de lagen

Een essentieel onderdeel van ArchiMate is de afstemming tussen de verschillende lagen. Er wordt niet afzonderlijk naar één laag gekeken maar naar het geheel. Het linken van de verschillende lagen en de communicatie tussen deze is relevant en sluit aan op mijn bevindingen van het eerste hoofdstuk.



Figuur 12: Voorbeeld Business Application Alignment [BOSM05]



Figuur 13: Voorbeeld applicaties ondersteund door infrastructuur [BOSM05]

3.8 Mijn bevindingen

Op basis van dit hoofdstuk kan ik de volgende deelvraag beantwoorden:

- **deelvraag 3:** Hoe wordt er gecommuniceerd op enterprise-architectuur niveau?

Tijdens het bestuderen van deze modelleertaal, is de theorie uit het vorige hoofdstuk direct bevestigd. Dat enterprise-archtiectuur zich op een hoog niveau afspeelt, is duidelijk af te zien aan de voorbeeld-modellen. De specifieke functionaliteit van een applicatie, de inhoud van een bedrijfsproces of de technische kenmerken van hardware, zijn niet zichtbaar in een ArchiMate-model. Een ArchiMate-model geeft inzicht in de architectuur, in de samenhang van de verschillende lagen en de relaties tussen artefacten van deze lagen. Het belang van de samenhang en integratie van verschillende domeinen en architecturen, wordt door ArchiMate als inter-domein-relaties beschreven, waarbij er niet zo zeer gesproken wordt over domeinen en architecturen, maar over lagen. De taal biedt de mogelijkheid om een ontwerp te maken, specifiek voor één laag (business, applicatie of technologie), of een ontwerp waarbij één of meer van deze lagen samen geïntegreerd zijn. Hoogevorst maakt onderscheid in vier hoofdontwerpdomeinen waarbij elk domein deelarchitecturen bevat. Dit onderscheid wijkt enigszins af van het ArchiMate framework:

- Business hoofdontwerpdomein komt overeen met de ArchiMate businesslaag.
- Technologie hoofdontwerpdomein bevat de technologie architectuur maar ook de IT architectuur, die voorschrijft hoe IT-systemen moeten worden ontworpen. Onder dit hoofdontwerpdomein behoort dus de ArchiMate applicatielaag en de technologielaag.
- Organisatie & informatie hoofdontwerpdomein wordt afgevangen door de horizontale dimensie van het ArchiMate framework. Hiertoe behoren namelijk de aspecten structuur, gedrag en informatie. Structuur en gedrag kan gedefinieerd worden als onderdeel van de organisatie.

Het framework bestaat uit 31 concepten en 12 relaties, een breed scala aan artefacten die gebruikt kunnen worden om een architectuur te visualiseren. De verschillende relaties bieden de mogelijkheid om structurele en dynamische relaties tussen de concepten te leggen. De structurering van dimensies biedt de mogelijkheid om een enterprise vanuit meerdere gezichtspunten (viewpoints) te modelleren. Dit is belangrijk aangezien de belangen van stakeholders een belangrijke invloed hebben op het ontwerp van een architectuur.

4 Requirements Management

4.1 Inleiding

In de inleiding wordt er gesproken over "voorafgestelde requirements". U beeld zich wellicht een lijst in, met per regel een eis, wens of een behoefte wat men graag wil terug zien in het ea-model. Heel abstract klopt deze gedachte, er komt echter wat meer bij kijken. Technieken en methoden met betrekking tot het achterhalen, definiëren en analyseren van requirements, wordt ook wel requirements management genoemd. Requirements management komt oorspronkelijk uit het softwareontwikkeling vakgebied. In de literatuur wordt er dus altijd gesproken over een systeem dat ontwikkeld of aangepast dient te worden. De vertaalslag van systeemniveau naar enterprise-architectuurniveau behandel ik op het einde van het hoofdstuk.

4.2 Definitie

Requirements betekent eisen en behoeften [VAND09] en spelen een belangrijke rol bij het ontwerp van systemen. De IEEE (Institute of Electrical and Electronics Engineers) beschrijft requirement als volgt:

- a condition or capability needed by a user to solve a problem or achieve an objective;
- a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed document;
- a documented representation of a condition or capability as in (1) or (2).

Finkelstein geeft de volgende definitie aan requirements management:

> **Requirements management** is the systems engineering activity principally concerned with finding, organizing, documenting and tracking requirements for software systems. Its focus is maintaining traceability, defined as the ability to describe and follow the life of a requirement, in both a forwards and backwards direction. [FINK00, p. 1]

Uit de hierboven beschreven definitie concludeer ik dat requirements management uit twee verschillende concepten bestaat. Het vinden, organiseren en documenteren, ook wel requirements engineering en het achterhalen (tracking) van requirements, requirements traceability.

4.3 Requirements Engineering

Requirements engineering is het proces om van een probleem tot een mogelijke oplossing te komen. Dit proces is relatief onafhankelijk van het ontwerpproces van het systeem. Het gaat er namelijk niet om hoe het ontworpen dient te worden, maar wat er ontworpen moet worden. Requirements engineering is het dichten van het gat tussen probleem en oplossing. Requirements engineering kan daarom opgesplitst worden in twee verschillende benaderingen, namelijk probleemgeoriënteerd en oplossinggeoriënteerd.

Probleemgeoriënteerde requirements engineering is het onderzoeken, achterhalen en vaststellen van het probleem. Dit gebeurt voordat men gaat kijken naar een oplossing. Oplossinggeoriënteerde requirements engineering bestaat uit het ontwerpen en beschrijven van het systeemgedrag en hiervoor mogelijke oplossingen te presenteren. Bray combineert deze twee benaderingen:

> **Requirements engineering** is investigating and describing a problem domain and requirements and designing and documenting the characteristics for a solution system that will meet these requirements. [BRAY02, p. 23]

4.3.1 Fases requirements engineering

Blaauboer verdeelt het requirements engineering proces in zeven verschillende fases, namelijk: inception, elicitation, analyses, negotiation, specification, validation en verification. [BLAA06, p. 4]

4.3.1.1 Inception

Het requirementsproces begint met een project inception. Inception staat letterlijk voor "aanvang" [VAND09] en houd in dat er een beslissing binnen de organisatie is genomen om een IT oplossing te ontwikkelen. Deze beslissing is gemaakt op basis van bijvoorbeeld de strategie of financiële aspecten. [BLAA06, p. 4]

4.3.1.2 Elication

Elication betekent ontlokken en loskrijgen [VAND09]. Tijdens deze fase zoekt men uit welke informatie nodig is, welke bronnen gebruikt worden en welke technieken worden toegepast. De nodige informatie bestaat uit achtergrondinformatie over het probleemdomein, beperkingen die opgelegd zijn door de cliënt, de omgeving en technologie, maar ook de problemen die daadwerkelijk een oplossing

vereisen. In deze fase is de probleemgeoriënteerde benadering duidelijk aanwezig. Andere bronnen zoals technische standaarden, bestaande systemen of specificaties en documenten binnen het domein zijn tevens relevant. [BLAA06, p. 4]

4.3.1.3 Analyses

Wanneer de elication fase is afgerond, wordt deze verzamelde informatie geanalyseerd. In deze fase worden (in abstracte wijze) de requirements duidelijk. Activiteiten binnen deze fase zijn modelleren en controleren. Zowel het systeem als het domein wordt gemodelleerd. [BLAA06, p. 5]

4.3.1.4 Negotiation

De requirements die het resultaat zijn van de analysefase, zullen in eerste instantie voor een aantal conflicten zorgen. Wellicht is er incomplete informatie, zijn niet alle stakeholders tevreden of wordt het budget voor het ontwikkelen van het systeem overschreden. Deze fase wordt in de praktijk vaak gecombineerd met de analysefase. De elication, analyses en negotiation fase wordt in een iteratief proces uitgevoerd totdat men overeenstemming bereikt over de requirements. Dit onderlinge verband wordt in figuur 2.2.1 weergegeven. Wanneer requirements met elkaar botsen, is overleg met de stakeholders van cruciaal belang. [BLAA06, p. 5]

4.3.1.5 Specification

In deze fase zijn de requirements nog vrij ruw en globaal en dienen deze nu gespecificeerd te worden. Hiervoor heeft de IEEE (830) een standaard ontwikkeld. Deze standaard helpt bij het genereren van een document, waarin het vereiste gedrag (de gespecificeerde requirements) van het systeem beschreven staat. [BLAA06, p. 5]

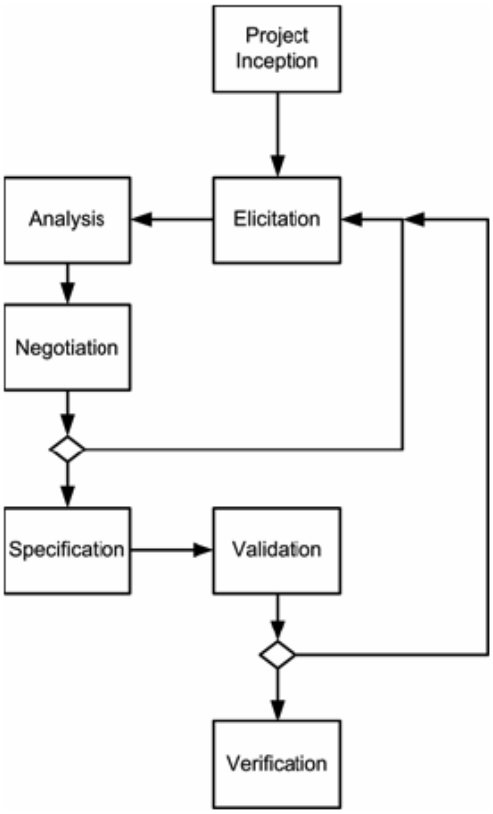
4.3.1.6 Validation

Wanneer de requirements onjuist geformuleerd zijn, zal er een defect systeem ontwikkeld worden. Het is daarom van belang dat de gespecificeerde requirements van de specification fase worden gecontroleerd. Het controleren op compleetheid en consistentie wordt ook wel requirements validation genoemd. Wanneer er sprake is van onjuist gespecificeerde requirements, gaat men terug naar de elication fase. Dit iteratieve proces wordt afgebeeld in het onderstaande figuur en zal net zo lang herhaald worden totdat de specificaties foutloos zijn. Wanneer deze fase is voltooid is er een volledig functioneel systeemontwerp en kan men beginnen met het implementeren van het systeem. [BLAA06,

p. 6]

4.3.1.7 Verification

Wanneer het systeem is ontwikkeld, is er nog één stap over binnen het requirements engineerings proces, namelijk de verification fase. In deze fase worden de requirements gecontroleerd in het ontwikkelde systeem. Dit gebeurt vaak in de acceptatietest waarbij de systeemfunctionaliteit één op één wordt gecontroleerd met de requirements.



Figuur 14: Requirements Engineering Proces Framework [BLAA06]

Bij elke fase behoren activiteiten en technieken die in de onderstaande tabel worden weergegeven. In verband met de scope van deze Master Thesis worden deze verder niet toegelicht.

	Hoofd activiteiten	Technieken
Inception	Identify stakeholders, record their desires, identify constraints, structure requirements, identify existing systems and derive requirements from them, investigate architectural requirements.	Interviews, surveys, groupsessions, process model analysis, organizational documents analysis, goal identification, viewpoints, introspection, scenarios, stakeholder identification, prototyping, protocol analysis, investigate business events
Elaboration/Analyse	Check for necessity, check for completeness and consistency, check for feasibility	Structured analysis, goal analysis, object oriented analysis, problem framing, scenario analysis, checklists
Negotiation	Discuss the requirements with all stakeholders, prioritize the requirements, agreeing on the requirements	EMAP, SIBYL, ROI calculation, pair wise comparisons, 100 dollar techniques, QFD, voting schemes, win-win, Prioritization workgroups
Specification	Transform objectives into system functions, Specify behavior, Write the specification	Document writing techniques, modeling techniques (ERD, dataflow, goal modeling, context diagrams, etc)
Validation	Review specification, develop prototypes, write manuals, write test cases, perform acceptance tests	Simple checks, document reviews, formal validation, prototyping, scenarios, functional testing, writing manuals.

Tabel 6: Activiteiten en technieken requirements engineerings proces

4.4 Requirements Traceability

Traceability betekent letterlijk opspoorbaarheid, de mogelijkheid om iets terug te voeren [VAND09]. Requirements traceability is bedoeld om een eindproduct te herleiden tot aan de requirements en dat de requirements op hun beurt herleid kunnen worden tot hun oorsprong (bijvoorbeeld stakeholders, concerns, documenten etc.). Een veel geciteerde definitie is die van Finkelstein:

> "**Requirements traceability** refers to the ability to describe and follow the life of a requirement, in both a forwards and backwards direction (i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases)." [FINK94, p. 1]

Requirements traceability kan op twee verschillende niveaus worden toegepast, namelijk: tussen de bronnen en de requirements en tussen de requirements en het eindproduct. David onderscheid vier verschillende vormen van requirements traceability. [DAVI90]:

- **Backward-from-requirements traceability**

  De herkomst van elke requirement is te herleiden tot aan stakeholders, documenten, systeem requirements etc.

- **Forward-from-requirements traceability**

  Elke requirement is tot (een deel van) een component of meer in het eindproduct te herleiden.

- **Backward-to-requirements traceability**

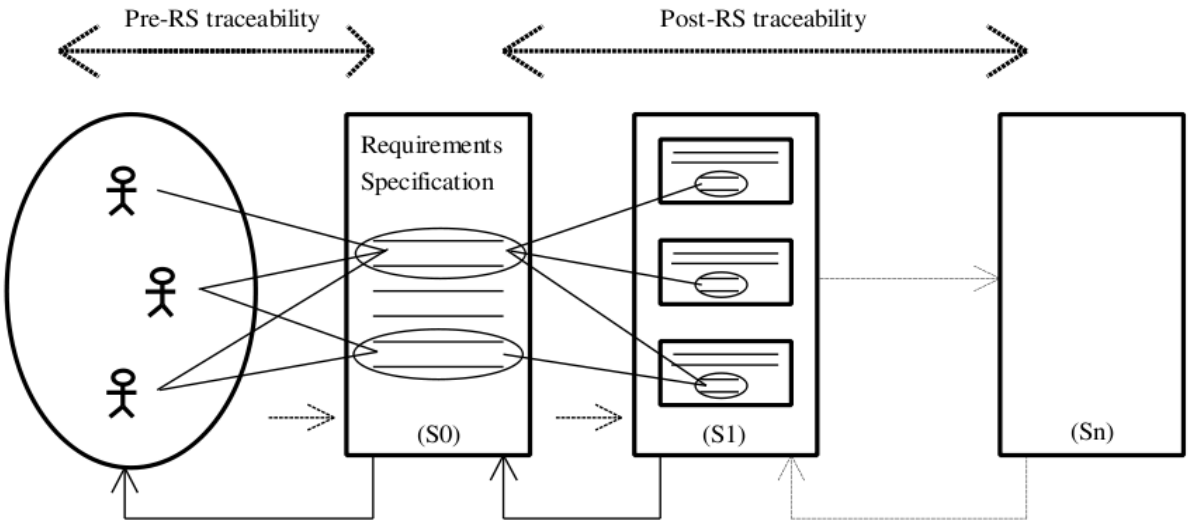
  Elke component in het eindproduct is te herleiden tot een of meer requirements.

- **Forward-to-requirements traceability**

  Iedere bron kan herleid worden naar één of meer requirements.

Na de voltooiing van de requirements specificatie, is backward-from-requirements tracebility en forward-to-requirements traceability van belang. Wanneer het eindproduct getest wordt, zijn forward-from-requirements traceability en backward-to-requirements traceability belangrijk.

Finkelstein maakt onderscheid tussen twee verschillende types van requirements traceability, namelijk *pre-rs traceability* en *post-rs traceability*. Dit word in het onderstaande figuur schematisch weergegeven:



Figuur 15: Basis types voor requirements traceability [FINK94, p. 11]

Pre-RS traceability is het herleiden van de requirements vanuit of naar de originele bronnen zoals stakeholders en documenten. Het gaat hier om het proces voordat de requirements in de specificatie zijn opgenomen [FINK94, p. 10, 11]. Pre-RST omvat backward-from-requirements traceability en forward-to-requirements traceability.

Post-RS traceability bestaat uit de activiteiten die de levensloop van requirements omvatten, na de requirementsspecificatie. Het concretiseren van de requirements in de specificatie wordt als uitgangspunt genomen om tot het eindproduct te komen [FINK94, p. 10, 11]. Post- RST omvat forward-from-requirements traceability en backward-to-requirements traceability.

4.5 Type requirements

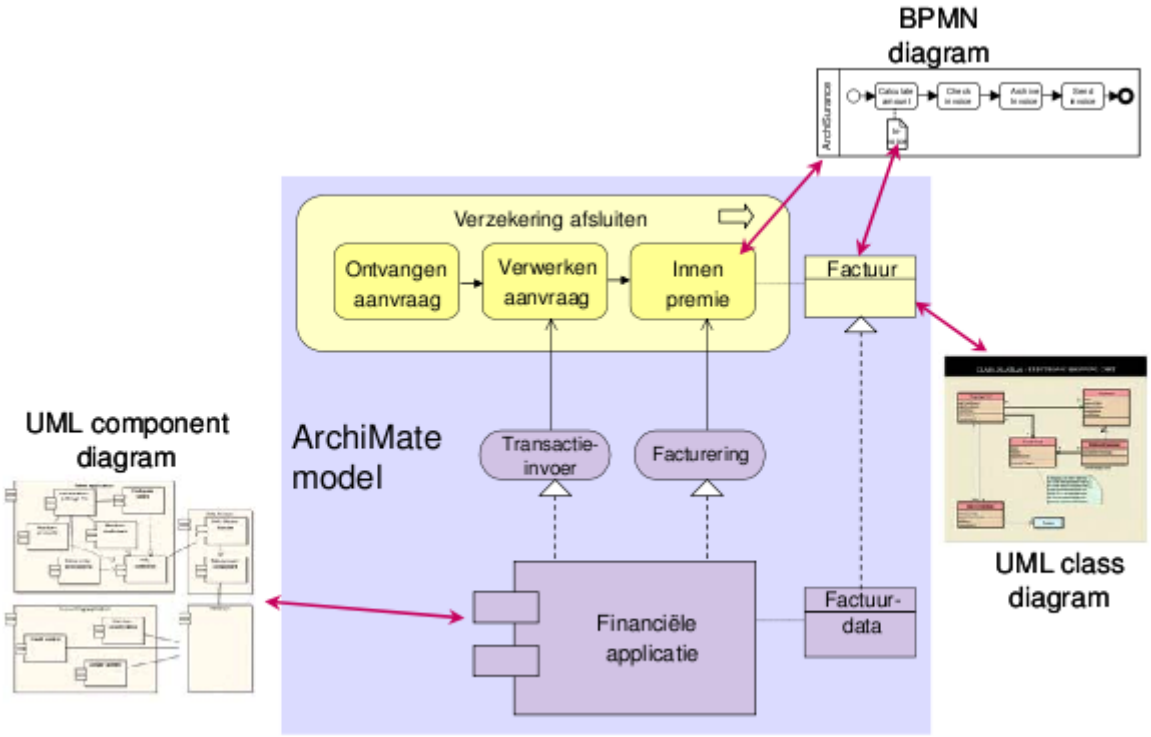
Enterprise-architectuur zorgt voor een globale benadering op gebied van requirements management. De requirements worden niet gespecificeerd voor individuele systemen, maar voor meerdere gelinkte systemen in verschillende domeinen en deelarchitecturen die samen een gemeenschappelijk doel hebben. Requirements over het specifieke gedrag en functionaliteit van een systeem of applicatie, bevinden zich op een lager niveau, ik noem dit voor het gemak, het operationele niveau:

“Binnen softwareontwikkeling wordt vaak het onderscheid gemaakt tussen de enterprise-wereld en de systeem-wereld. De enterprise-wereld beschrijft het domein waarin het softwaresysteem een bepaalde service aanbiedt, terwijl het systeem-domein beschrijft wat het systeem precies moet doen, met de

daarbij behorende beschrijvingen van de systeemeisen, conceptuele modellen en implementaties." [LOUC95]

ArchiMate beschrijft niet de details die zich op een lager niveau binnen de verschillende domeinen afspelen [THEO092]. Er zijn namelijk deeloplossingen voor de verschillende architectuurterreinen beschikbaar, zoals Business Process Modeling Notation (BPMN) voor gedetailleerde beschrijvingen van een bedrijfsproces of Unified Modelling Language (UML) voor softwareontwikkeling.

Deze richten zich echter op een gedetailleerde beschrijving van een beperkt deel van een enterprise-architectuur en niet op de hoofdlijnen en de samenhang van de verschillende gebieden zoals ArchiMate dit doet. Lankhorst legt uit [LANK04] dat het koppelen van deze modellen mogelijk is.



Figuur 16: ArchiMate-model als brug tussen andere modellen [LANK04]

4.6 Mijn bevindingen

Op basis van dit hoofdstuk kan ik de volgende deelvragen beantwoorden:

- **deelvraag 4:** Op welke manier is het mogelijk om requirements te herleiden?

Requirements management bestaat uit requirements engineering en requirements traceability. Requirements engineering bestaat uit zeven verschillende fases, waarbij de fases specification en validation betrekking hebben op mijn onderzoek. De overige fases doelen op het ontlokken, loskrijgen en achterhalen van de requirements. Dit is in mijn onderzoek niet relevant aangezien ik onderzoek op welke manier het mogelijk is om de requirements te herleiden naar een ea-model, waarbij de requirements al bestaan en niet meer achterhaald dienen te worden.

Op basis van het onderscheid wat door Loucopoulos (1995) gemaakt wordt, onderscheid ik twee types requirements die betrekking hebben op mijn onderzoek: de requirements die gebaseerd zijn op de principes en de belangen van stakeholders: ook wel **architectuurrequirements** genoemd, en de requirements die opgesteld worden op basis van het architectuurontwerp voor het ontwerp van een systeem: **systeemrequirements**.

Als men kijkt naar de traditionele vorm van requirements management, praat men over requirements die gebaseerd zijn op een bepaalde functionaliteit van een individueel systeem. Dit is bij enterprise-architectuur niet het geval, aangezien deze requirements over het algemeen gelden voor meerdere gelinkte systemen in verschillende domeinen.

Enterprise-architectuur beschrijft hoe de organisatie, de informatievoorziening, de applicaties en de infrastructuur hun vorm hebben gekregen en hoe zij zich voordoen in het gebruik. Keuzes die gemaakt worden op architectuurniveau beïnvloeden systemen die zich op een lager niveau van een enterprise bevinden. Ik denk dat systemen op operationeel niveau ook invloed kunnen uitoefenen op de architectuur van een enterprise: wanneer het IT budget het niet toe laat om een nieuw systeem te ontwerpen of om het bestaand systeem te wijzigen, betekent dat men de functionaliteit van een systeem moet accepteren en dat deze systeembeperking invloed zal hebben op het ontwerp van de architectuur. Architectuur beïnvloed de functionaliteit van systemen, maar verouderde (legacy) systemen kunnen ook de architectuur beïnvloeden.

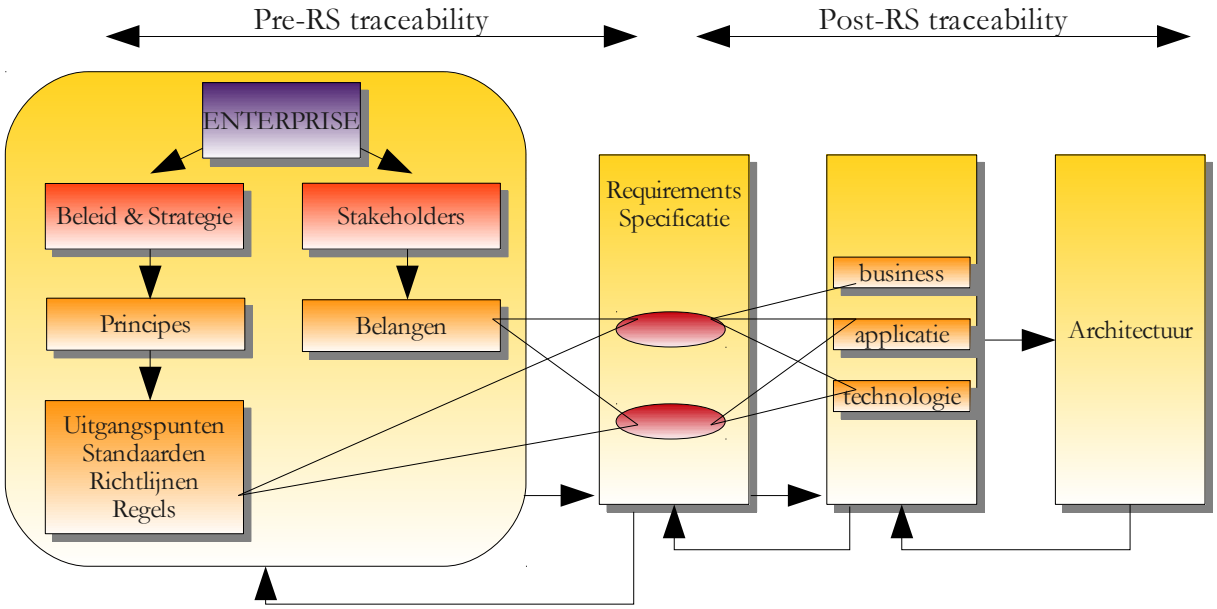
Een **requirementstaal** voor **enterprise-architectuur** moet de mogelijkheid hebben om requirements op architectuurniveau te kunnen herleiden naar requirements van systeemontwerpen. Hierdoor kan men zien wat voor een invloed een principe / strategisch doel heeft op de functionaliteit van een systeem en kan men zien wat voor een invloed een systeembeperking heeft op het ontwerp van een architectuur.

Zoals ik in mijn onderzoeksplan heb toegelicht, is compliance "het linken, herleiden en controleren van de overeenkomstigheid tussen een ea-model en de requirements zodat de integriteit van de architectuur wordt gewaarborgd en bevorderd." Het toetsen van deze compliance kan worden uitgevoerd, door middel van requirements traceability. Op basis van mijn bevindingen onderscheid ik twee requirements traceability processen:

4.6.1 Traceability-proces 1

Pre RS-traceability: het herleiden van de requirements vanuit of naar de uitgangspunten, standaarden, richtlijnen en regels die gebaseerd zijn op de principes van een enterprise.

Post RS-traceability: het herleiden van de requirements vanuit of naar het ontwerp van een domein, deelarchitectuur of het complete coherente architectuurontwerp.



Figuur 17: Requirements Traceability 1 [FINK94] aangepast

Dit proces maakt het mogelijk om de compliance tussen de bron en de architectuur te toetsen waarbij

het volgende mogelijk is:

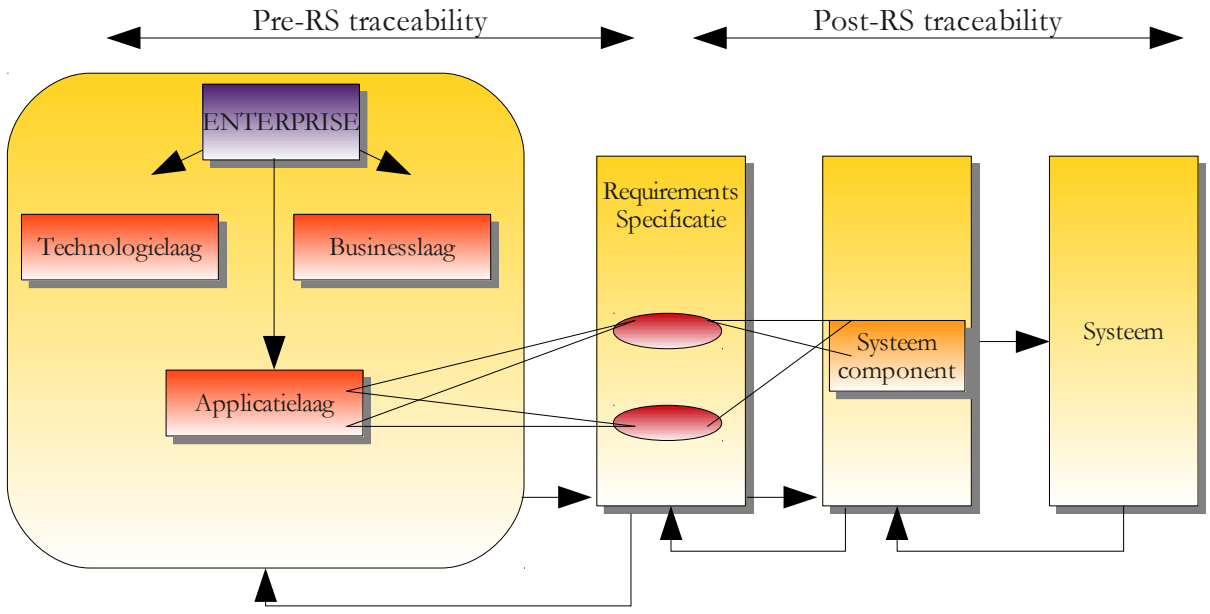
- Welke (architectuur)requirements zijn er ontstaan op basis van de principes of belangen en stroken deze met elkaar?
- Op basis van welke (architectuur)requirements is de architectuur ontworpen en stroken deze met elkaar?
- Hoe hebben de principes en belangen het ontwerp van de architectuur bepaald?

Op deze manier kan men het architectuurontwerp onderbouwen en motiveren.

4.6.2 Traceability-proces 2

Pre RS-traceability: het herleiden van de systeemrequirements vanuit of naar de bron (architectuurontwerp).

Post RS-traceability: het herleiden van de systeemrequirements vanuit of naar het ontwerp van een applicatie component of systeemontwerp.



Figuur 18: Requirements Traceability 2 [FINK94] aangepast

Dit proces maakt het mogelijk om de compliance tussen de bron en het systeem te toetsen waarbij het volgende mogelijk is:

- Welke systeemrequirements zijn er ontstaan op basis van het architectuurontwerp en stroken deze met elkaar?
- Op basis van welke systeemrequirements is het systeem ontworpen en stroken deze met elkaar?
- Hoe heeft het architectuurontwerp het ontwerp van het systeem bepaald?

Op deze manier kan men het systeemontwerp onderbouwen en motiveren.

Hoe past men Pre-RS en Post-RS traceability toe?

Ik ben van mening dat er een proces moet plaatsvinden om dit te kunnen toepassen. Ik denk dat een "belang" van een stakeholder, of een "principe" van een enterprise niet direct zichtbaar is in de requirementsspecificatie en dat men bepaalde stappen moet uitvoeren om tot een requirement te komen. **Er zal een proces plaats moeten vinden welke een abstract doel (principe, doelstelling, belang) operationaliseert naar een concrete requirement.** Op welke manier deze stappen uitgevoerd dienen te worden, ga ik in het volgende hoofdstuk onderzoeken.

5 Goal-oriented requirements engineering

5.1 Inleiding

Goal-driven requirements processing wil zeggen dat men requirements management uitvoert door de doelen die op het hoogste niveau van de enterprise zijn geformuleerd, als leidraad te gebruiken voor het uitvoeren van requirements management. Goal-driven requirements processing wordt in de literatuur meestal goal-oriented requirements engineering genoemd. Lamsweerde (2001) geeft de volgende definitie:

> **Goal-oriented requirements engineering** is het gebruik van doelen voor het uitlokken, uitwerken, structureren, specificeren, analyseren, onderhandelen, documenteren en wijzigen van requirements. [LAMS01]

Het vastleggen van deze doelen vindplaats op verschillende abstractieniveaus: doelen op hoog niveau (high-level goals) of doelen op laag niveau (low-level goals). Doelformaliseringen verwijzen naar de juiste eigenschappen die gewaarborgd moeten worden; het zijn uitspraken die optatief zijn, in tegenstelling van indicatieve uitspraken en zijn begrensd door het onderwerp [ZAV07]. Doelen kunnen betrekking hebben op functionele belangen (services die worden aangeboden) of niet-functionele belangen (kwaliteit van de service zoals bijv. beveiliging, performance etc.) van een systeem. Dit onderscheid ziet men ook bij de traditionele vorm van softwareontwikkeling.

Een requirement en een doel lijken veel op elkaar. Het verschil zit hem in het feit, dat een doel bereikt kan worden wanneer er verschillende actoren (agenten) en/of systemen worden gebruikt. [DARD93]

Een belangrijk onderdeel tijdens het proces van requirements engineering is het maken van keuzes, welk deel van het systeem wordt geautomatiseerd en welk deel niet. Wanneer het systeem ervoor zorgt dat het doel wordt behaald, is er sprake van een requirement. Wanneer de omgeving ervoor zorgt dat het doel wordt behaald, dan noemt men dit een aanname. Dit is een belangrijk onderscheid binnen het requirements engineering vak. In tegenstelling bij een requirement, kan een aanname niet worden afgedwongen door het systeem dat nog ontwikkeld dien te worden, deze wordt namelijk behaald door het organisatorische beleid van de organisatie [LAMS01]. De praktijk wijst uit dat verschillende systemen met verschillende requirements, gemeenschappelijke doelen kunnen hebben. Verschillende requirements kunnen dus één of meerdere doelen ondersteunen.

5.2 Relevantie

Er zijn verschillende redenen waarom het gebruik van doelen relevant is tijdens het requirements engineerings proces [LAMS01]:

- Het opstellen van een complete specificatie van de requirements is zeer belangrijk. Doelen geven nauwkeurige criteria aan voldoende compleetheid. De specificatie is compleet wanneer er bewezen kan worden dat alle doelen behaald kunnen worden met de eigenschappen van het beschouwde domein.
- Het vermijden van irrelevante requirements is tevens een belangrijk onderdeel. Een requirement is onbetwistbaar wanneer er bewezen kan worden dat deze tenminste één doel realiseert.
- Het uitleggen en verklaren van requirements aan de stakeholders. Een requirement bestaat omdat deze is voortgekomen uit één of meerdere doelen. Door het doel uit te werken door middel van een doelverfijning (goal-refinement), is het mogelijk om een hoger-niveau doel te herleiden naar een lager-niveau doel (bijvoorbeeld naar technische requirements), en vice versa.
- Doelverfijning zorgt voor een natuurlijk mechanisme waarmee complexe requirements gestructureerd uitgewerkt kunnen worden.
- Alternatieve doelverfijning (alternitave goal refinements) zorgt ervoor dat er eventuele alternatieven gemodelleerd kunnen worden. Het maken van beslissingen en keuzes wordt hiermee ondersteund.
- Wanneer requirements eventueel met elkaar conflicteren, is het mogelijk om alternatieven voor te stellen.
- Het scheiden van stabiele en vluchtige informatie. Een requirement representeert een manier om een doel te behalen. Een requirement kan zich ontwikkelen en kan ook op een andere manier dit doel vervullen. Dit is bij een doel niet het geval. Hoe hoger een doel zich bevindt, des te stabieler het is. De praktijk wijst uit dat verschillende systemen met verschillende requirements, dezelfde doelen kunnen behalen.
- Doelen zorgen voor de identificatie van requirements. Doelen zijn samen met scenario's de basis voor het uitvoeren van het analyseproces (zie voorgaand hoofdstuk).

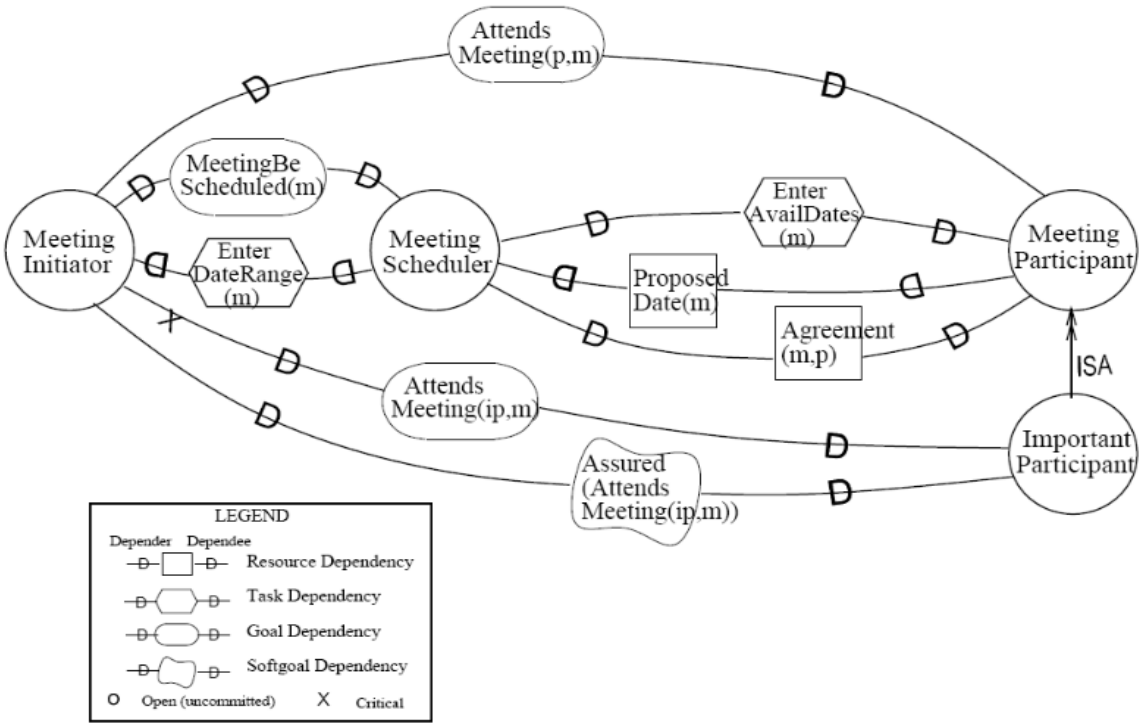
5.3 Talen

Twee bekende goal-oriented requirementstalen zijn het I* Framework en KAOS. Deze twee talen zal ik in het kort toelichten.

5.3.1 I* Framework

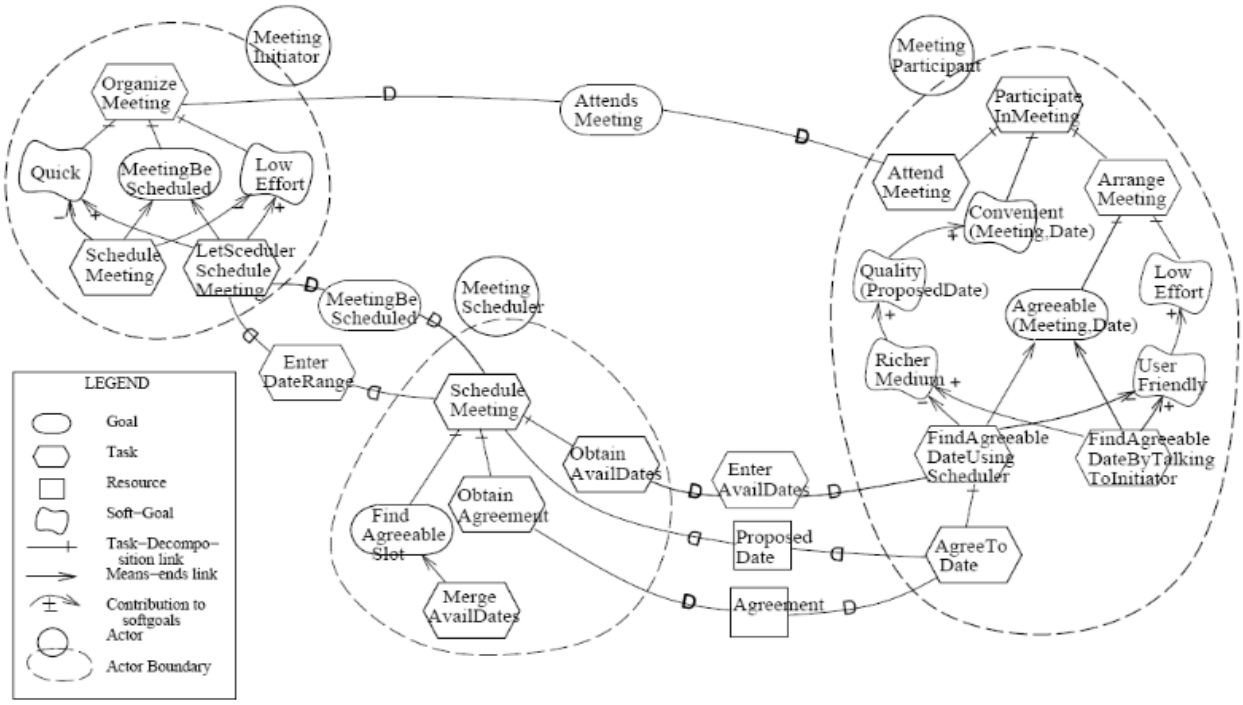
Het I* Framework focust zich op concepten bij het modelleren en analyseren tijdens een vroeg stadium van het requirements engineering proces (probleemgeoriënteerde requirements engineering). Het accent ligt op het "waarom" en niet zo zeer op het "wat". Dit framework is ontwikkeld om de organisatorische omgeving en de bijbehorende informatiesystemen te modelleren en hierover te redeneren. Het begrip dat bij deze taal centraal staat is de intentionele actor (intentional actor). Actoren binnen een organisatie hebben intenties en eigenschappen zoals doelen, overtuiging, talent en toewijding. Actoren rekenen op andere actoren zodat bepaalde doelen behaald kunnen worden, taken kunnen worden uitgevoerd en andere bronnen gebruikt kunnen worden. Actoren zijn strategisch en proberen deze afhankelijkheden te schikken zodat men kan omgaan met kansen en bedreigingen.

Het I* Framework onderscheid twee verschillende modellen, namelijk het Strategic Dependency (SD) Model en het Strategic Rationale (SR) Model. Het SD Model beschrijft de afhankelijkheden tussen verschillende actoren binnen een organisatorische context. Een afhankelijkheid modelleert een overeenkomst tussen twee actoren, waarbij de ene actor (the depender) afhankelijk is van de andere actor (the dependee) om een doel te kunnen bereiken, om een taak uit te kunnen voeren of om een bron te gebruiken / aan te kunnen leveren. Een afhankelijkheid kan een zacht doel (soft goal) bevatten. Bij een zacht doel is het onduidelijk aan welke criteria het moet voldoen om het doel te kunnen bereiken. Een zacht doel is daarom slecht meetbaar. Een hard doel (hard goal) is precies en meetbaar gedefinieerd. Onderstaande figuur illustreert een voorbeeld van een SD model:



Figuur 19: Voorbeeld Strategic Dependency Model [QUAR092, p. 12]

Het SR Model beschrijft de interesses en belangen van een stakeholder en hoe deze opgevraagd kunnen worden door middel van verschillende configuraties van de omgeving en systemen. Het SR Model voegt meer detail toe aan het SD Model door de te kijken naar de interne relaties. Intentionele elementen zoals doelen, taken en bronnen komen zowel bij interne als externe afhankelijkheden voor. Intentionele elementen kunnen gelinkt worden door means-ends relaties en taak decomposities. Een derde type link is de contributierelatie (contribution relation), welke aangeeft hoeveel een doel of taak bijdraagt aan een zacht doel. Het I* Framework bied vier verschillende types en niveaus van analyse: asses the ability, workability, viability, believability of goals en tasks. Onderstaande figuur illustreert een voorbeeld van een SR model:



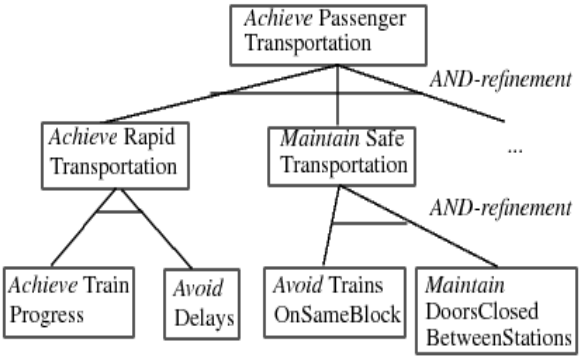
Figuur 20: Voorbeeld Strategic Rationale Model [QUAR092, p. 13]

5.3.2 KAOS

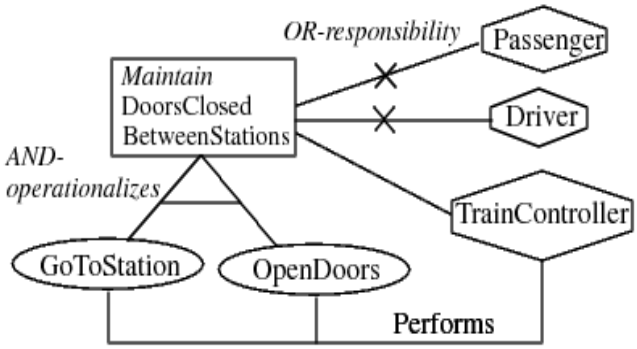
In tegenstelling tot het I* Framework richt KAOS zich meer op het latere stadium van de requirements fase (oplossinggeoriënteerde requirements-engineering). In KAOS is het tevens mogelijk om de motivatie te modelleren (waarom), echter houdt deze taal zich minder bezig met het modelleren van de intentionele actoren. Net zoals het I* Framework, is het concept "doel" belangrijk, Met een doel verwijs ik hier naar een gewenst effect/toestand in het probleemdomein of gewenste eigenschappen van een oplossing. Een doel wordt behaald wanneer er coöperatie tussen agenten plaatsvindt. Een agent kan een actor zijn die betrokken is bij het volbrengen van een doel, maar ook bestaande informatie, een applicatie of een menselijke gebruiker.

Doelen bestaan op verschillende abstractieniveaus, waarbij er onderscheid wordt gemaakt tussen hoger-niveau doelen (higher-level goals) en lager-niveau doelen (lower-level goals). Hoger-niveau doelen worden gerealiseerd door de lager-niveau doelen, wat door middel van een verfijningsrelatie gemodelleerd wordt. Dit type relatie wordt ook gebruikt om aan te geven waarom bepaalde lager-niveau doelen nodig zijn. Een hoger-niveau doel vereist de coöperatie van meerdere systemen.

Wanneer een doel behaald wordt en deze status ervoor zorgt dat een ander doel niet behaald kan worden (en vice versa), is er een sprake van conflict. Dit wordt gemodelleerd door middel van een conflictrelatie. Deze relatie geeft aan welke doelen samen voor een conflict zorgen. Een obstakel (hindernis, obstacle) wordt gebruikt om een situatie te modelleren die ervoor zorgt dat een bepaald doel niet behaald kan worden. Deze hindernis zou opgelost kunnen worden wanneer overige doelen behaald worden. KAOS biedt verder de mogelijkheid om eigenschappen van het probleemdomein te modelleren, zoals domeinhypotheses en domeininvarianten. KAOS ondersteund verschillende analysemethoden zoals traceability, compleetheid, formele validatie, verfijningscontrole, risicoanalyses, bedreigingsanalyses en conflictanalyses.



Figuur 21: Voorbeeld KAOS model (goal refinement)



Figuur 22: Voorbeeld KAOS model (operationalization & responsibility)

5.4 Mijn bevindingen

Op basis van dit hoofdstuk kan ik de volgende deelvraag beantwoorden:

- **deelvraag 5:** Aan welke eisen moet een requirementstaal voor enterprise-architectuur voldoen?

Goal-oriented requirements engineering is van wezenlijk belang voor enterprise-architectuur en bezit een aantal zeer nuttige eigenschappen. Lamsweerde (2001) geeft hier een aantal argumenten voor, waaronder: het zorgen voor voldoende compleetheid en het vermijden van irrelevante requirements, welke belangrijk zijn de probleemgeoriënteerde fase van het requirements management proces. Ik denk dat de volgende concepten belangrijk zijn voor mijn onderzoek:

Doelverfijning

Doelverfijning zorgt ervoor dat requirements gestructureerd uitgewerkt kunnen worden. Ik denk dat dit proces relevant is, met name voor het toetsen van de compliance omdat men op deze manier kan controleren of er overeenkomstigheid is tussen een principe, doel of een belang van een stakeholder en een requirement (pre-rs traceability) of tussen een requirement en een component/complete architectuur van een ea-model (post-rs traceability). Op deze manier is het mogelijk om requirements te herleiden en te verklaren aan stakeholders en kan men controleren of er tussen beiden sprake is van volledige compliance.

Alternatieve doelverfijning en conflictrelaties

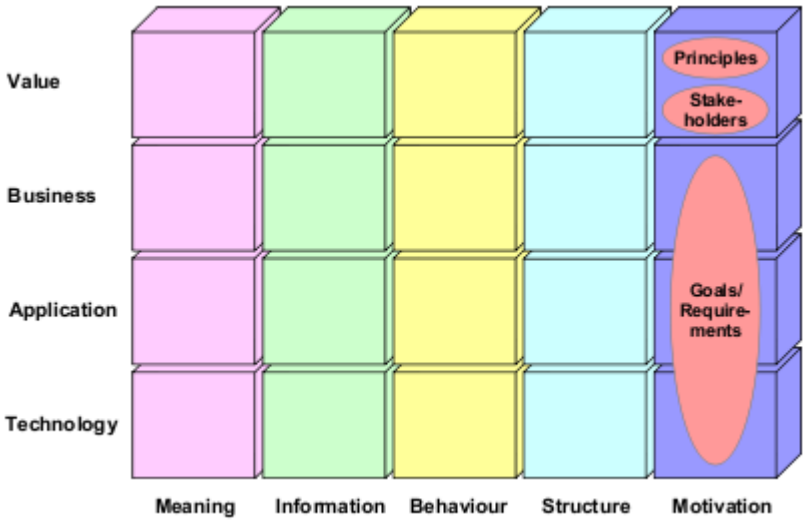
Door alternatieven te modelleren kan men verklaren waarom er bepaalde keuzes en beslissingen zijn gemaakt. In sommige gevallen kunnen keuzes onduidelijk zijn en wordt er niet gekozen voor de meest vanzelfsprekende optie. Dit kan bijvoorbeeld komen omdat er tussen requirements een conflict aanwezig is of omdat er een aantal beperkingen in de architectuur aanwezig zijn welke niet direct in het model te herleiden zijn. Deze conflicten worden gemodelleerd door middel van de conflictrelatie. Een eventueel alternatief wordt gemodelleerd door middel van een alternatieve doelverfijning.

6 ARMOR: requirementstaal voor EA

6.1 Inleiding

ARMOR is ontwikkeld door Novay (voorheen Telematica Instituut), BiZZdesign en de Universiteit van Twente. Deze taal heeft als doel om de onderliggende motivatie (principes, belangen van stakeholders en requirements) van enterprise-architectuur te ondersteunen. Deze taal is gebaseerd op bestaande requirements modelleer-technieken (KAOS en I*Framework, zie vorig hoofdstuk) en is tevens compatibel met ArchiMate. Op basis van mijn literatuuronderzoek, ga ik in het tweede deel van deze Master Thesis ARMOR toetsen en onderzoeken, waarbij ik hopelijk een aantal zwakke plekken aan het licht kan brengen. Ik hoop daarna een advies/opzet voor een eventuele uitbreiding of verbetering te presenteren. Dit hoofdstuk is gebaseerd op het artikel [QUAR091] en het bijhorend rapport [QUAR092] van het ARMOR onderzoek.

Quartel benadrukt dat er op gebied van enterprise-architectuur modellering, de focus meestal op “as is” en “to be” architecturen ligt (opgedeeld in structuur, gedrag en informatie). Quartel bevestigt hiermee de descriptieve en prescriptieve benadering die in het hoofdstuk enterprise-architectuur is behandeld. ARMOR legt de focus op het waarom, dus de onderliggende motivatie, reden, doelstellingen en requirements van een enterprise. Het ArchiMate-framework wordt door middel van ARMOR uitgebreid met een tweetal aspecten op de horizontale dimensie (meaning, motivation) en een extra laag op de verticale dimensie (value). Dit wordt in het onderstaande figuur weergegeven:



Figuur 23: ArchiMate-framework met ARMOR [QUAR091, p. 5]

Het motivatieaspect (motivation aspect) houdt zich bezig met de doelen en intenties van de enterprise en bestaat uit doelen en requirements die gebaseerd zijn op de principes en de concerns (belangen) van de stakeholders. Het betekenisaspect (meaning aspect) representeert de concerns die gerelateerd zijn aan de semantiek van de enterprise-architectuur-artefacten. Naast deze nieuwe aspecten wordt de waarde (value*)* laag toegevoegd. Deze laag representeert de waarde van de services en producten die aan de klanten worden aangeboden. Deze laag wordt gebruikt om specifieke type waarden (bijvoorbeeld kosten) te modelleren.

In het gepubliceerde onderzoek wordt alleen het motivatie aspect behandeld, de aspecten betekenis en waarde worden buiten beschouwing gelaten. Deze twee aspecten worden hoogstwaarschijnlijk binnenkort in een nieuw artikel gepresenteerd. De ontwikkelaars van ARMOR hadden de volgende eisen voor ogen waaraan de taal moest voldoen:

- Hergebruik van concepten en ideeën van bestaande requirements-modelleertalen
- Compatibel met ArchiMate.
- Mogelijkheid om documentatie, communicatie en redenering van requirements vast te leggen.
- Eenvoudig te gebruiken. Makkelijk te begrijpen, te leren en toe te passen.
- Het moet mogelijk zijn om ARMOR uit te breiden met gespecialiseerde concepten en analysetechnieken. De gebruiker kan op deze manier kiezen voor een basis of een geavanceerde versie van ARMOR.
- Het moet mogelijk zijn om abstracte doelstellingen te herleiden naar concrete doelstellingen en ontwerpartefacten (zoals services en producten die deze abstracte doelen implementeren).

Het motivatieaspect bevat drie domeinen die door ARMOR aan ArchiMate worden toegevoegd, namelijk het stakeholder-, principles- en requirementsdomein.

Voordat men is begonnen met de ontwikkeling van de taal, is er uitgebreid onderzoek gedaan naar bestaande requirements-talen. Op basis van deze resultaten zijn er keuzes gemaakt die de formele definitie van de taal hebben bepaald. Verschillende concepten uit het Business Motivation Model, I* Framework en KAOS zijn gebruikt om tot de definitie van de ARMOR taal te komen. De verantwoording en motivatie van deze keuzes zijn te lezen in de eerste twee hoofdstukken van [QUAR092].

6.2 De concepten

6.2.1 Doel

Het sleutelconcept van ARMOR (net zoals bij de het I*Framework en KAOS) is “doel”. Voorbeelden zijn “tevreden klant”, “de verkoop verbeteren”, “het calculeren van de wisselkoers”, “afleveren van bepaalde goederen”. In het algemeen kunnen deze doelen gerealiseerd worden door de samenwerking van meerdere systemen, of door de samenwerking van systemen die nog ontwikkeld moeten worden (system-to-be). Er kan onderscheid gemaakt worden tussen harde en zachte doelen.

6.2.2 Requirement

Een requirement is een (type) doel dat toegewezen wordt aan een bepaald systeem, zodat het systeem verantwoordelijk is voor het behalen van het doel. Een verwachting (expectation) wordt gezien als een requirement die gekoppeld is aan een systeem dat nog gebouwd dient te worden (system-to-be). Een requirement overerft de eigenschappen van een doel. Een doel kan gebaseerd zijn op één of meerdere verwachtingen. Het expliciet maken van deze verwachtingen wordt aangeraden.

6.2.3 Relaties

Een doel kan verfijnd worden in sub-goals. Er zijn drie verschillende verfijningen:

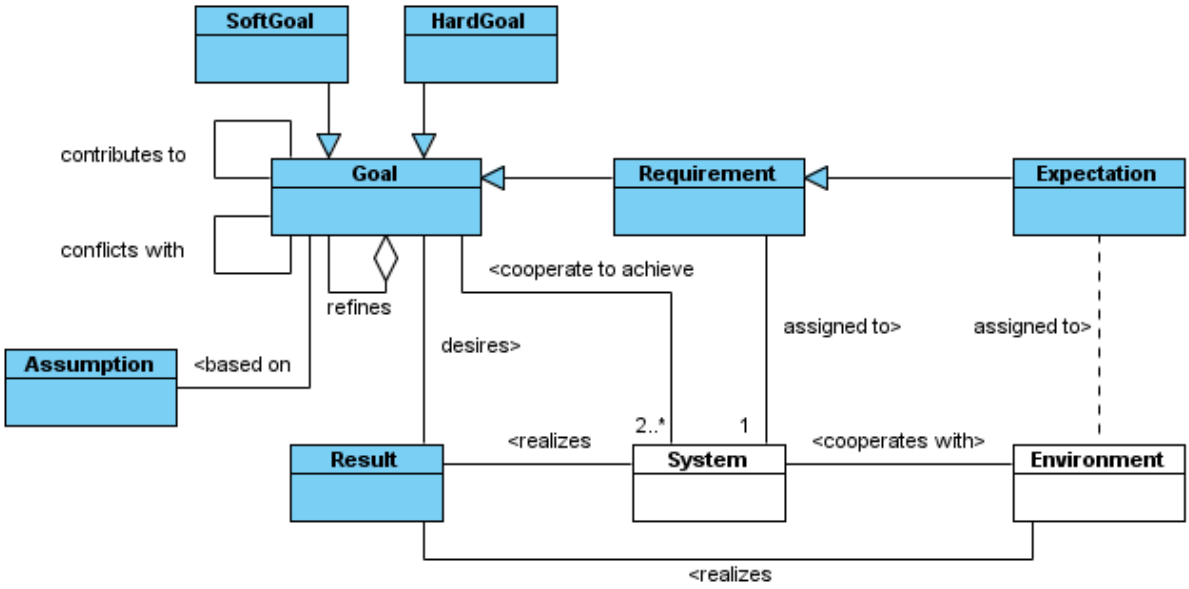
- AND-refinement (conjunction): alle sub-doelen moeten worden behaald om te voldoen aan het (hoofd)doel;
- OR-refinement (disjunction): ten minste één doel moet worden behaald zodat het (hoofd)doel behaald wordt.
- Een combinatie van beiden: meestal in disjunctieve normaal vorm (een disjunctie van conjuncties).

Het is mogelijk dat een doel positief of negatief bijdraagt aan een ander doel. Dit gebeurt door middel van een contributierelatie. Voorbeeld: we nemen aan dat G2 een hard doel is, dan zou het kunnen zijn dat G1 (en G1 is geen sub-doel van G2) positief bijdraagt aan de criteria die zijn vastgesteld voor G2. Eigenschappen van deze relatie zijn:

- type: het type van de bijdrage (positief of negatief);
- strength: De sterkte van de bijdrage (zwak of sterk).

ARMOR maakt gebruik van een conflictrelatie. Bijvoorbeeld wanneer G1 conflicteert met de voldoening van G2 (G1 en G2 zijn beiden een hard doel). Deze conflictrelatie is symmetrisch; G1 conflicteert met G2 en G2 conflicteert met G1.

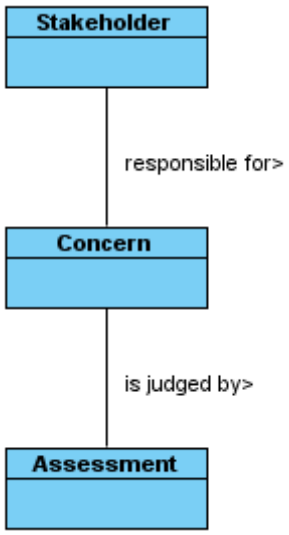
Hieronder wordt het conceptueel model van ARMOR weergegeven. De concepten system en environment zijn in het wit gemodelleerd omdat deze geen deel uitmaken van de taal. Deze zijn toegevoegd om meer duidelijkheid te creëren.



Figuur 24: Conceptueel model ARMOR [QUAR092, p.23]

6.3 Stakeholder-domein

Dit domein modelleert per architecturale laag de stakeholders van de enterprise, hun concerns (belangen) en de assessment (beoordeling) van deze belangen. Een belang is een gebied van aandacht of interesse. Stakeholders en hun belangen worden geïdentificeerd uit een businessplan of op basis van functies en verantwoordelijkheden binnen een enterprise. Zowel de gebruiker als een klant van een enterprise, kan gezien worden als een stakeholder. Belangen van stakeholders zullen met regelmaat beoordeeld worden. Hiermee kan de behoefte worden vastgesteld.



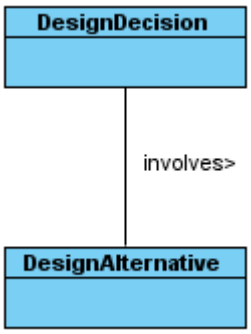
Figuur 25: Stakeholders domein [QUAR092, p.29]

6.4 Principles-domein

Dit domein modelleert de visie, missie, strategie, beleid, principes en richtlijnen van de enterprise, overeenkomend met de hoger-doel beperking/restricties voor het ontwerp van de architectuur. Het ARMOR-onderzoek maakt duidelijk dat dit domein meer precisie nodig heeft en dat de relaties beter gedefinieerd dienen te worden. Het domein heeft de behoefte om de visie, missie, strategieën, beleid, principes en richtlijnen beter te definiëren, echter is er nog geen duidelijke richtlijn hoe dit gerealiseerd moet worden. BMM zou hierin behulpzaam kunnen zijn.

6.5 Rationale-domein

Dit domein modelleert de beslissingen die het ontwerp van de enterprise inperken (design decisions). Hierbij behoren alle beslissingen die gemaakt zijn op basis van het businessplan, de verfijning van doelen, subdoelen en requirements, tot aan de implementatie van de bedrijfsservices, bedrijfsprocessen, applicatieservices en applicatiefuncties. Het modelleren van deze ontwerpkeuzes kan worden toegepast op elke ontwerpstrategie: top-down, bottom-up of middle-up. Het is ook mogelijk om ontwerpalternatieven te modelleren.



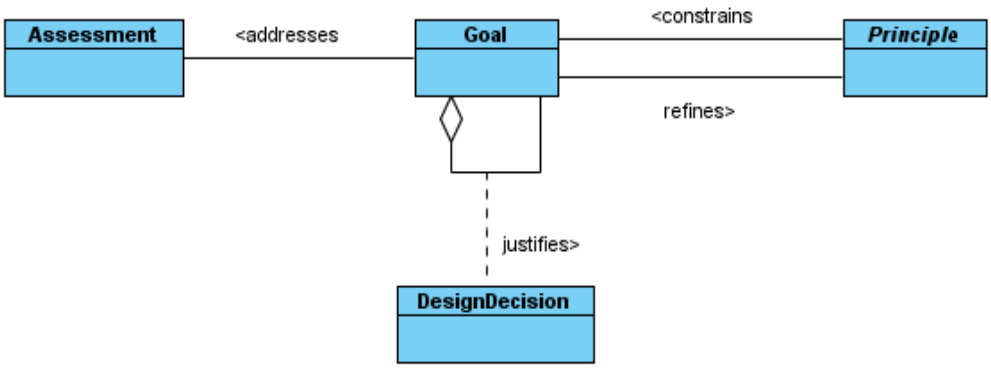
Figuur 26: Rationale domein [QUAR092, p.30]

6.6 Requirements-domein

Dit domein modelleert de doelen, requirements en verwachtingen die het ontwerp van de architectuur inperken. Deze komen meestal voort uit het principles-domein en de beoordelingen van de belangen uit het stakeholders-domein. Deze beoordelingen worden gedefinieerd door middel van sterke punten, zwakte punten, kansen en bedreigingen, die gebruikt worden om nieuwe doelen te stellen of bestaande aan te passen. De concepten van het requirements domein zijn in het conceptueel model zichtbaar, behandeld in de paragraaf "concepten".

Het requirements domein kan eventueel nog uitgebreid worden met twee subdomeinen, namelijk het use-case domein en business rules domein. Deze worden onder de paragraaf "uitbreidingen" behandeld.

6.6.1 Relaties met overige domeinen

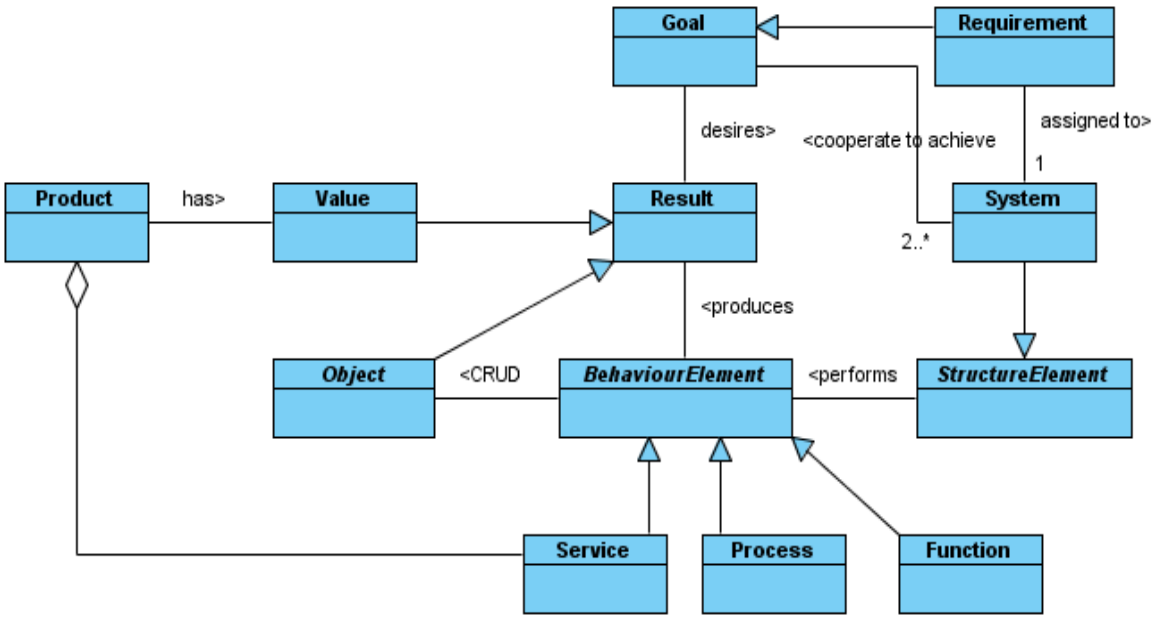


Figuur 27: Relaties requirementsdomein en overige domeinen [QUAR092, p.31]

- Een doel kan gelinkt zijn met één of meer beoordelingen van stakeholder concerns;
- De verfijning van een doel naar sub-doelen wordt gerechtvaardigd door een ontwerpkeuze;
- Een doel wordt beperkt en/of verfijnd door een principe.

6.6.2 Relaties met ArchiMate aspecten

In het onderstaande figuur wordt de relatie tussen het requirements-domein en de verticale dimensie van ArchiMate (structuur, gedrag, informatie) weergegeven:



Figuur 28: Relatie requirementsdomein en ArchiMate [QUAR092, p.31]

De abstracte concepten Object, BehaviourElement en StructureElement zijn gebruikt om domeinconcepten te representeren, die het structuur, gedrag en informatie aspect van een enterprise modelleren.

- Een requirement kan gekoppeld worden met het StructureElement (via het systeemconcept). Voorbeelden van structuurelementen zijn: organisaties, afdelingen, systemen, applicatie componenten etc.
- Een doel wordt dus behaald door de samenwerking van verschillende structuurelementen.
- Het resultaat dat wordt gewenst door een doel wordt geproduceerd door een BehaviourElement. Voorbeelden van deze gedragselementen zijn: business services, business

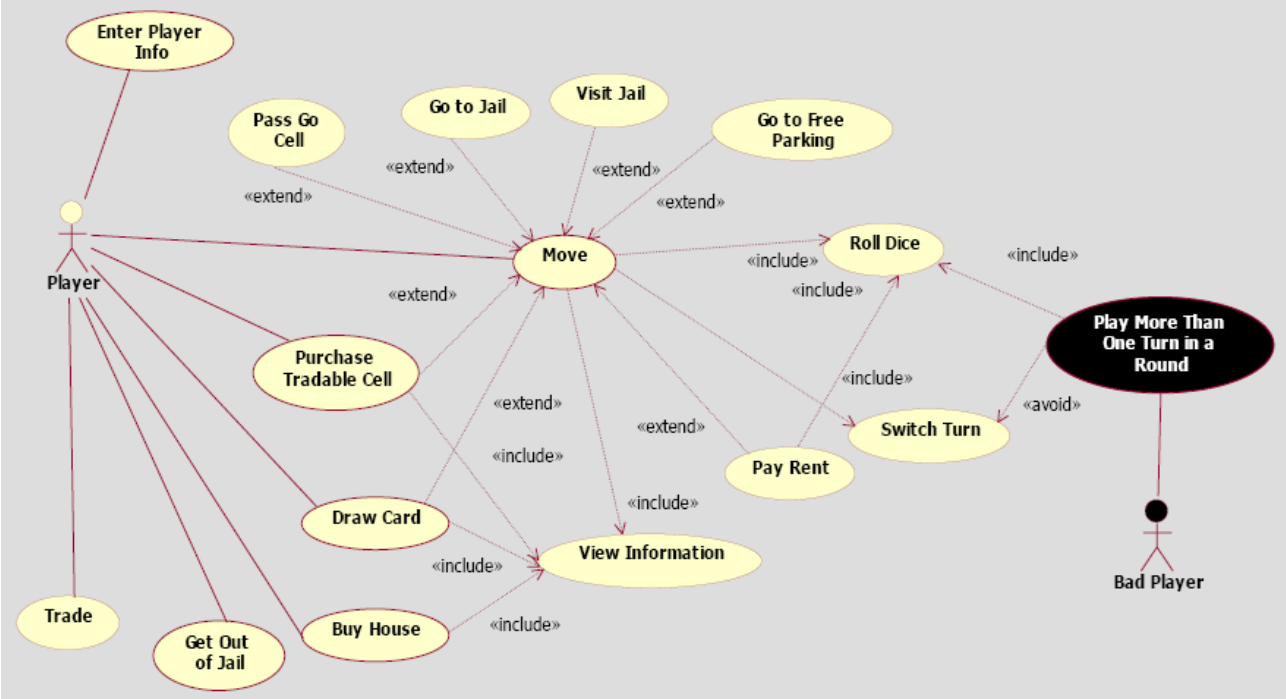
processes, applicaties services etc.

- Het resultaat dat wordt gewenst door een doel kan gerepresenteerd worden door een Object of een Value. Voorbeeld: De verkoop van een verzekering resulteert in de waarde "verzekerd".

6.7 Uitbreidingen

6.7.1 Use-case-domein

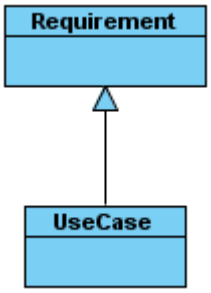
Het modelleren van use-cases lijkt erg veel op het modelleren van doelen en requirements. Use-cases worden voornamelijk gebruikt bij het vaststellen en het specificeren van requirements. Een use-case beschrijft de interactie tussen een systeem en de externe actor (gebruiker). Een use-case heeft altijd een bepaald doel voor ogen. Wanneer de use-case succesvol is uitgevoerd, is het doel behaald. Een use-case kan meerdere opeenvolgende scenario's beschrijven die fouten afhandelen of uitzonderingen weergeven. Een use-case wordt gedefinieerd als een requirementstype.



Figuur 29: Voorbeeld use case diagram [QUAR092, p.31]

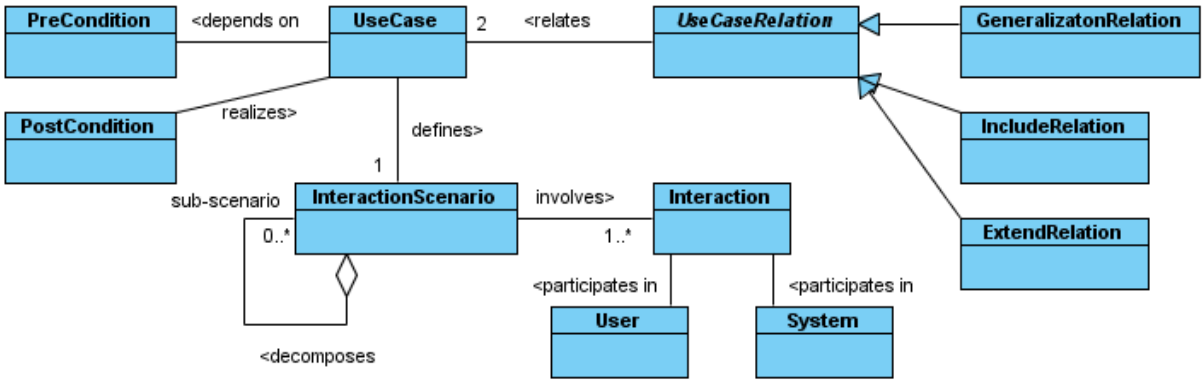
De interacties van een use-case definieert zowel de eigenschappen van het gedrag (functionaliteit) wat van het systeem gewenst is, alsmede het gedrag van de gebruiker (actor). Een use-case modelleert dus de requirements en de verwachtingen, welke aansluit op de filosofie van ARMOR.

Deze requirements en verwachtingen kunnen gezien worden als een verfijning van een doel of van een requirement. Een use case wordt voornamelijk gebruikt om systeemrequirements te definiëren, waarbij het mogelijk is om een requirement te verfijnen tot een abstracte systeemrequirement. Een use-case wordt gezien als een type requirement:



Figuur 30: Use-case als een type requirement [QUAR092, p.35]

Het onderstaande figuur geeft het conceptueel model van het use-cases-domein weer, welke een subdomein van het requirements-domein betreft:



Figuur 31: Use cases domein [QUAR092, p.35]

6.7.2 Business rules subdomein

The Business Rules Group definieert een bedrijfsregel op de volgende manier:

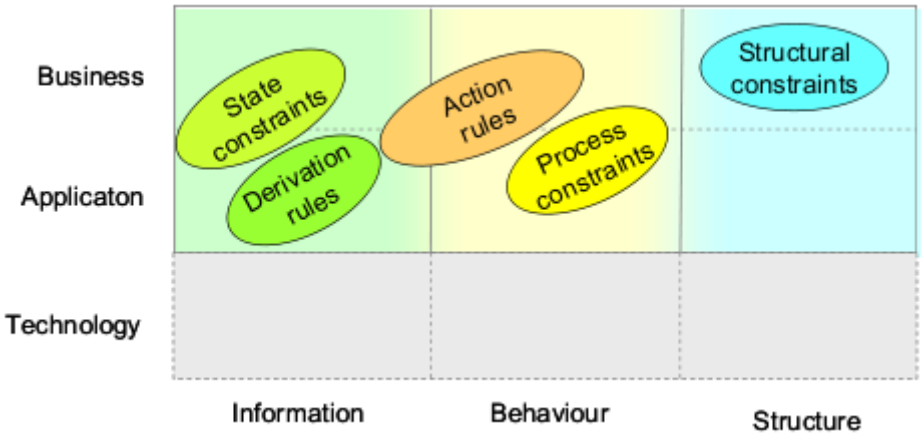
> A statement that defines or contrains some aspect of the business. It is intended to assert business structure or to control or influence the behaviour of the business".

ARMOR maakt onderscheid tussen twee type bedrijfsregels, namelijk een bedrijfsregel als een

ontwerpartefact of een bedrijfsregel als een requirement.

Bedrijfsregel als een ontwerpartefact

Het onderstaande figuur positioneert een bedrijfsregel als een ontwerpartefact binnen de business- en applicatielaag. Een bedrijfsregel wordt gemodelleerd als een deeloplossing voor het bedrijven van business binnen een enterprise. Wanneer men gebruik maakt van een bedrijfsregel taal zoals SBVR (Semantics of Business Vocabulary and Business Rules, OMG 2008), is het mogelijk dat deze regels worden geïnterpreteerd door een rule-engine. Verfijning van de bedrijfsregel is dan niet meer nodig.

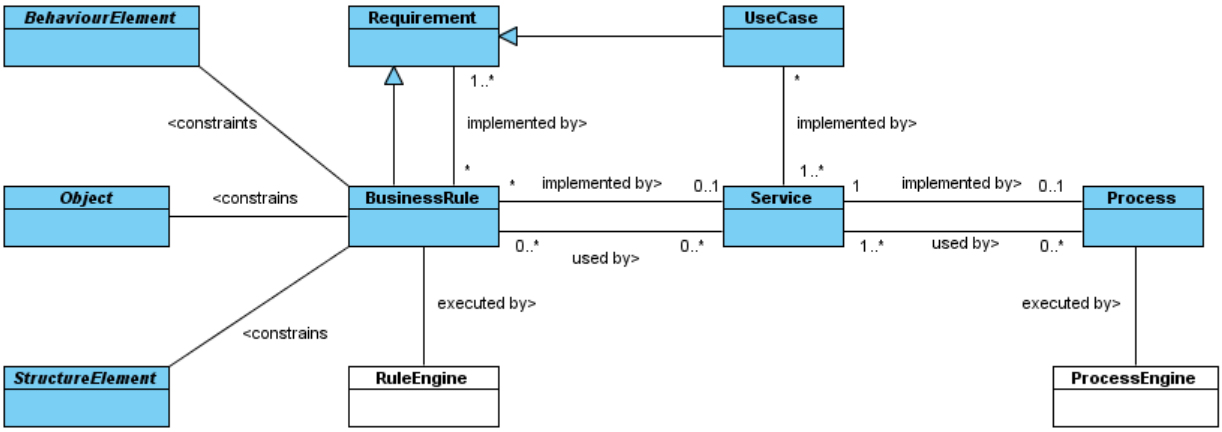


Figuur 32: Business rules in het ArchiMate Framework [QUAR092, p.36]

Bedrijfsregel als een requirement

Een bedrijfsregel kan ook gezien worden als een requirement, de inrichting van de bedrijfsprocessen van een enterprise. Een bedrijfsregel is meestal uitvoerbaar en kan bestaan uit meerdere uitvoerbare acties. Wanneer de bedrijfsregel bestaat uit meerder uitvoerbare acties, is verfijning noodzakelijk voordat men deze kan automatiseren.

Het bedrijfsregel concept overerft de eigenschappen van een requirement. De relatie tussen het business domein en de overige domeinen wordt in het onderstaande figuur weergegeven:

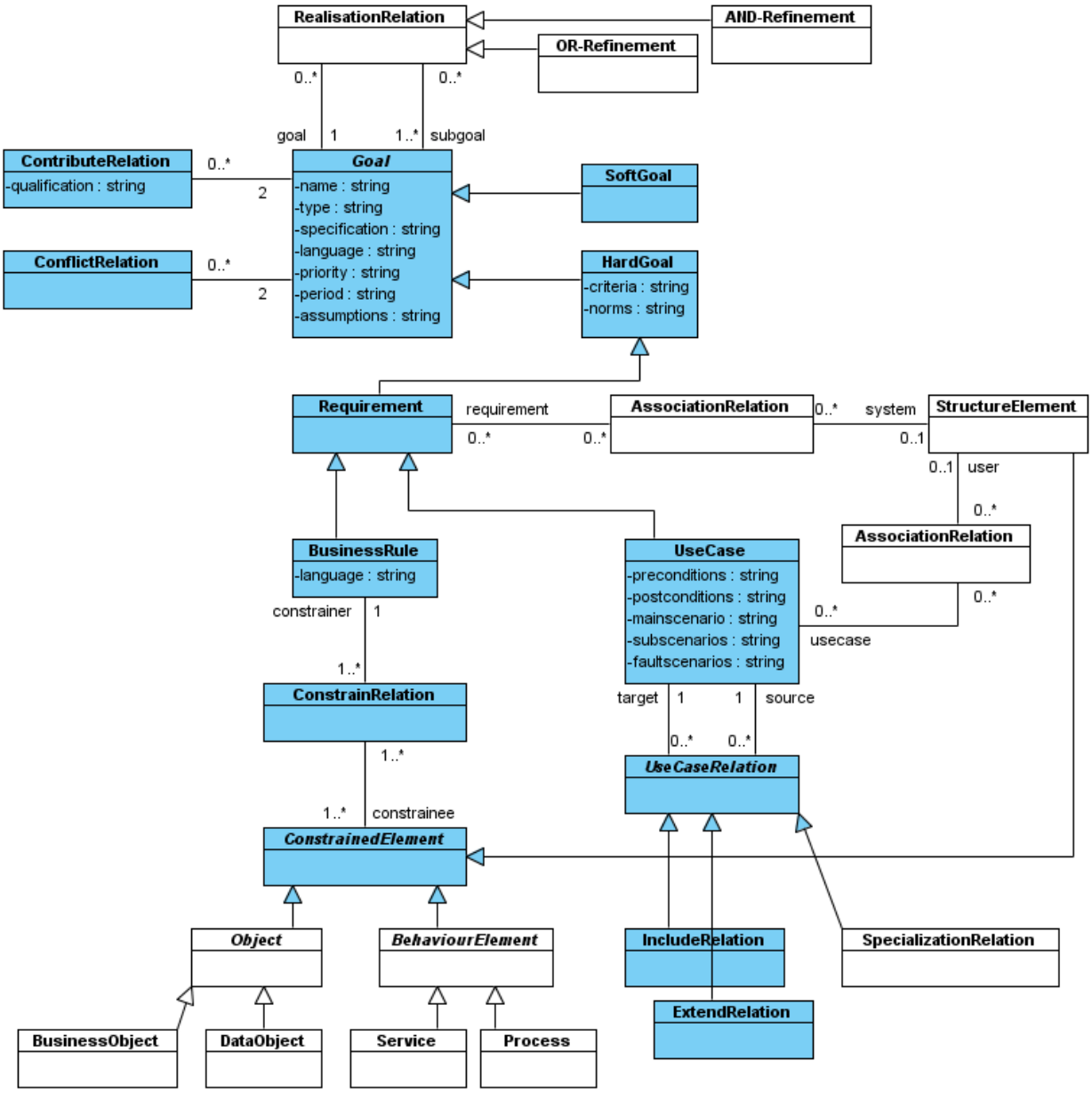


Figuur 33: Relatie business-rulesdomein en overige domeinen [QUAR092, p.38]

6.8 Definitie van de taal

6.8.1 Meta-model

Hieronder volgt een UML klassendiagram welke het meta-model van ARMOR definieert.



Figuur 34: Abstracte syntax van ARMOR [QUAR092, p.40]

De witte klasses zijn geen onderdeel van ARMOR, maar zijn taalelementen van andere modelleerdomeinen (o.a. ArchiMate).

Voor de volgende concepten zijn keuzes gemaakt:

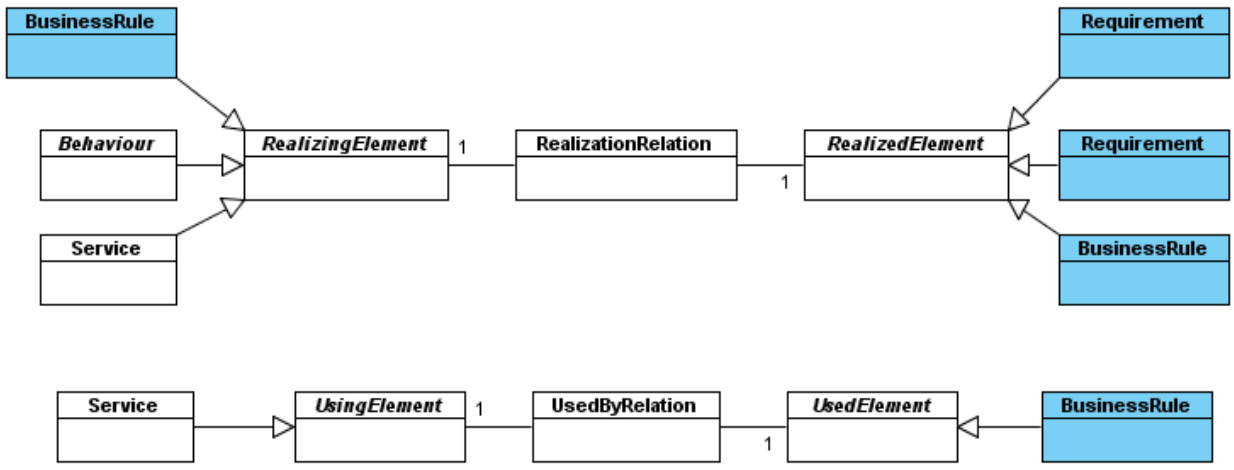
- De refinementrelatie tussen een goal en een sub-goal is gekoppeld met de "realization" relatie van ArchiMate;
- De "assigned to" relatie tussen een requirement en het "structure" element is gekoppeld met de "association" relatie van ArchiMate;
- Aannames worden weergegeven door het "assumptions" attribuut van het goal element;
- Het resultaat van een goal wordt weergegeven door het "specification" attribuut. Dit attribuut wordt gebruikt om het doel zo precies mogelijk te beschrijven, inclusief het resultaat;
- Er is geen taalelement gespecificeerd voor het "expectation" concept. Dit concept wordt nog niet ondersteund.

6.8.2 Uitbreiding realization & used by relatie

Het onderstaande model geeft de abstracte syntax weer van de "realization" en "used by" relatie. De "realization" relatie wordt bepaald door:

- een requirement en een bedrijfsregel;
- een requirement en een behaviour element, zoals een service of proces;
- een service en een bedrijfsregel.

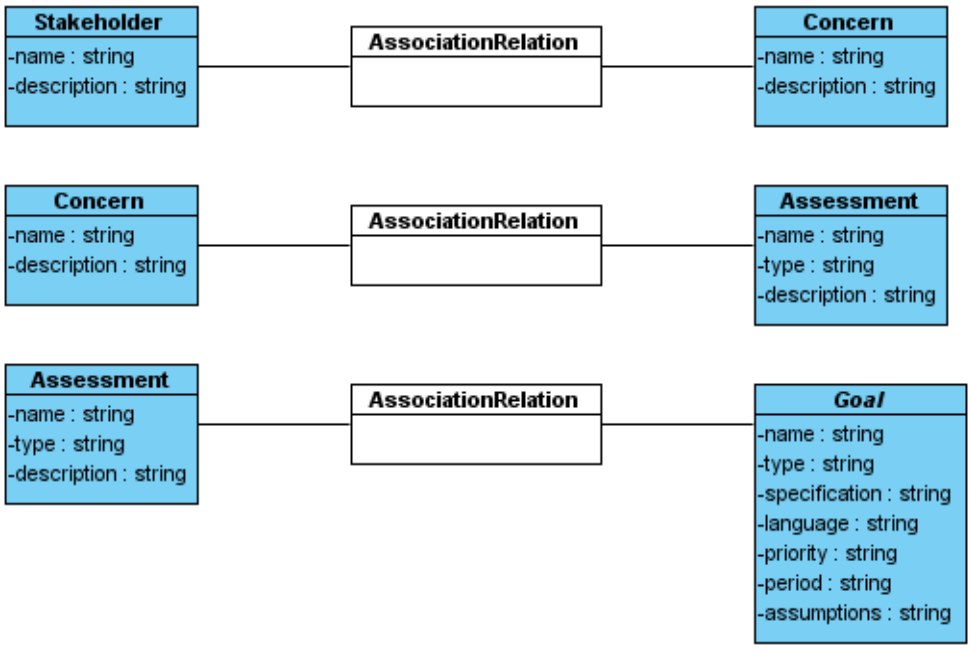
De "used by" relatie wordt bepaald tussen een service en een bedrijfsregel .



Figuur 35: Uitbreiding van de "realization" en "used by" relaties [QUAR092, p.41]

6.8.3 Uitbreiding met stakeholder-domein

Het stakeholder-domein is optioneel, hieronder de abstracte syntax van dit domein:



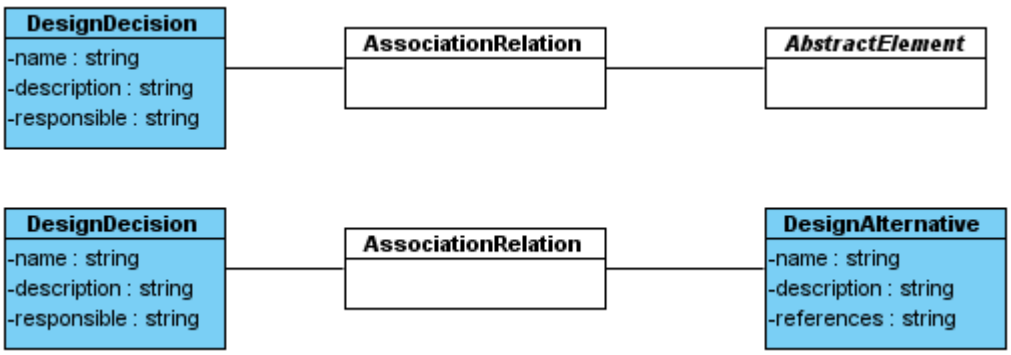
Figuur 36: Abstracte syntax van het stakeholders domein [QUAR092, p.43]

De "association" relatie van ArchiMate wordt voor het volgende gebruikt:

- de "responsible for" relatie tussen stakeholder en zijn belang;
- de "is judged by"relatie tussen een belang en de bijbehorende beoordeling;
- de "addresses" relatie tussen een beoordeling en doel.

6.8.4 Uitbreiding met het rationale-domein

Het rationale-domein is tevens optioneel, hieronder de abstracte syntax van dit domein:



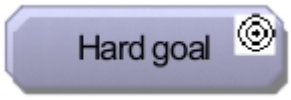
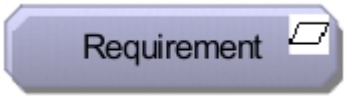
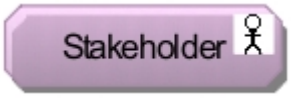
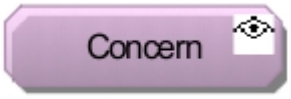
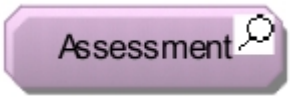
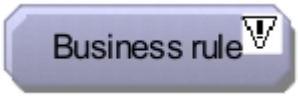
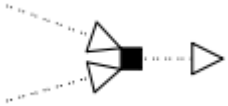
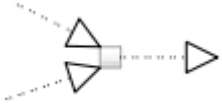
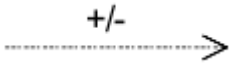
Figuur 37: Abstracte syntax van het rationale domein [QUAR092, p.43]

De bestaande “association” relatie wordt voor het volgende gebruikt:

- de “involves” relatie tussen de ontwepkeuzes en het ontwerpalternatief;
- de “justifies” relatie tussen de ontwerpkeuzes en ontwerpelement.

Op de volgende pagina volgt een overzicht van de concrete syntax van ARMOR. Deze syntax wordt gebruikt voor de daadwerkelijke toepassing van de taal.

6.8.5 Syntax

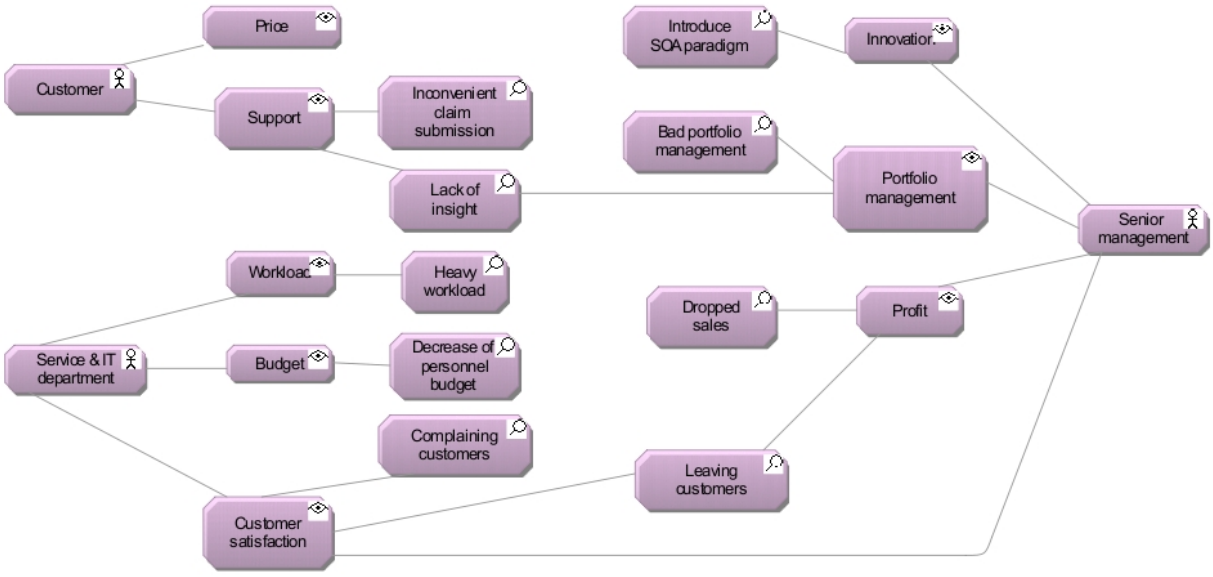
Element	Notatie
Hard doel	
Zacht doel	
Requirement	
Stakeholder	
Belang	
Beoordeling	
Use-case	
Bedrijfsregel	
Realisatierelatie	
AND realisatie	
OR realisatie	
Contributierealisatie	
conflictrelatie	
Includerelatie	
Extendrelatie	
Constrainrelatie	

Tabel 7: ARMOR Syntax

6.9 Voorbeelden

De introductie van ARMOR biedt voor enterprise-architectuur modellering de mogelijkheid om vanuit het requirements engineering domein te beredeneren en te analyseren. Hieronder worden een aantal voorbeelden gegeven door middel van een fictief verzekeringsbedrijf "PRO-Fit". In deze voorbeelden zijn de volgende stakeholders gemodelleerd: Klant, Service & IT afdeling en het Senior Management.

6.9.1 Stakeholders en hun belangen



Figuur 38: Stakeholder domein-model [QUAR092, p.46]

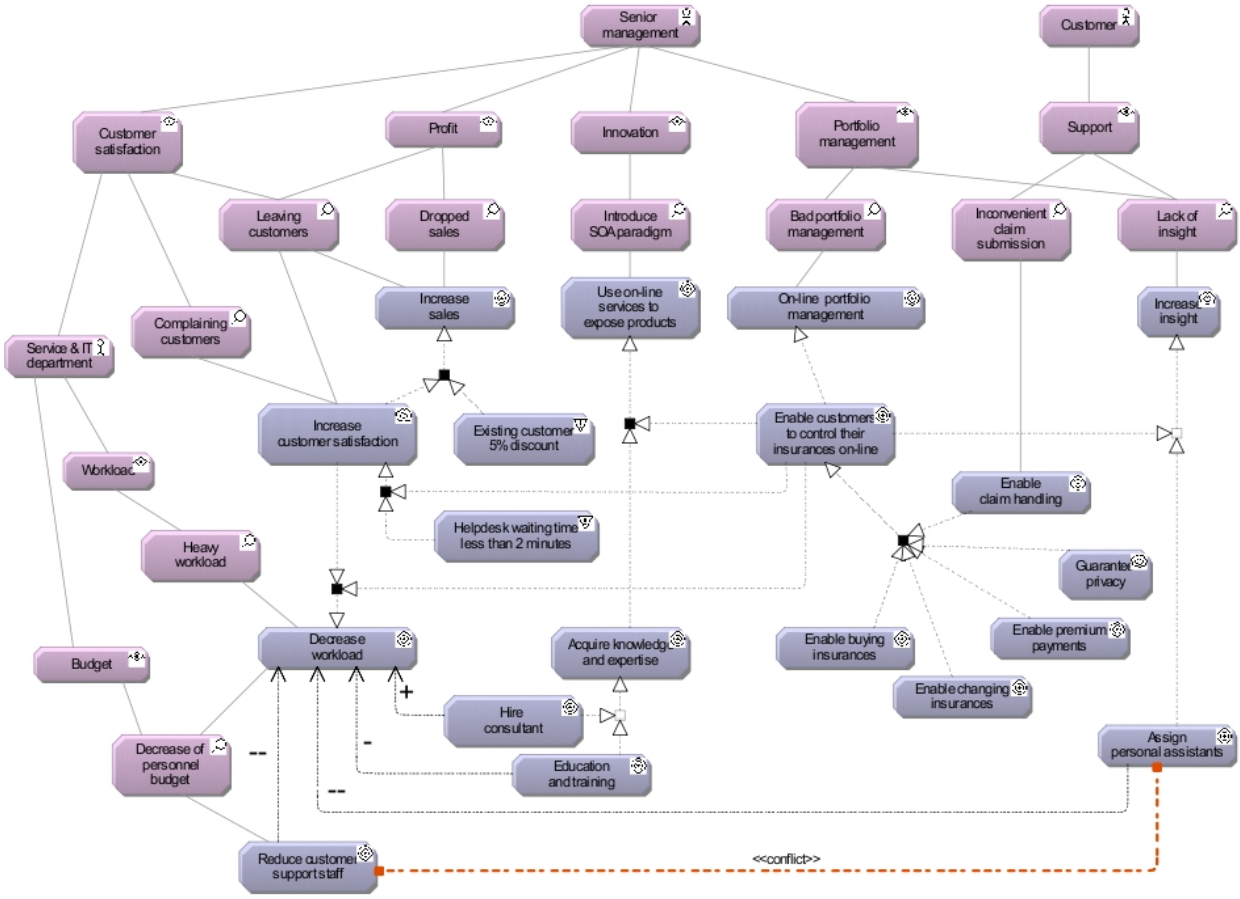
6.9.1.1 Beschrijving: Stakeholder domein-model

De klant heeft als belang dat zijn verzekering een aantrekkelijke prijs heeft en dat hij wordt ondersteund bij het aanpassen van zijn verzekeringsportefeuille. De Service & IT afdeling heeft als belang het ondersteunen van hun klanten, de werkdruk van de afdeling en hun budget voor klanten en IT gerelateerde activiteiten. Het Senior Management houdt zich voornamelijk bezig met de winst die het bedrijf moet maken, het managen van de klantenportefeuille, het initiëren van innovaties om winstgevend te blijven en net zoals de Service & IT afdeling, de klant tevreden houden.

Afhankelijk van het belang, wordt deze meerdere malen per jaar geëvalueerd. Tijdens deze evaluatie wordt er een beoordeling (assessment) toegewezen op basis van een SWOT (sterkte, zwakte, kansen,

bedreigingen) analyse. Het bovenstaande figuur bevat voornamelijk bedreigingen en zwaktes. De beoordeling van het belang "Support" resulteert in het feit dat klanten hun verzekeringsclaims via het internet willen indienen en dat ze niet content zijn met de huidige manier van beheren van hun verzekeringsportefeuille. De beoordeling van het belang "Profit" resulteert in het feit dat de verkoop is gedaald en dat bestaande klanten weggaan of geen nieuwe verzekeringen afsluiten.

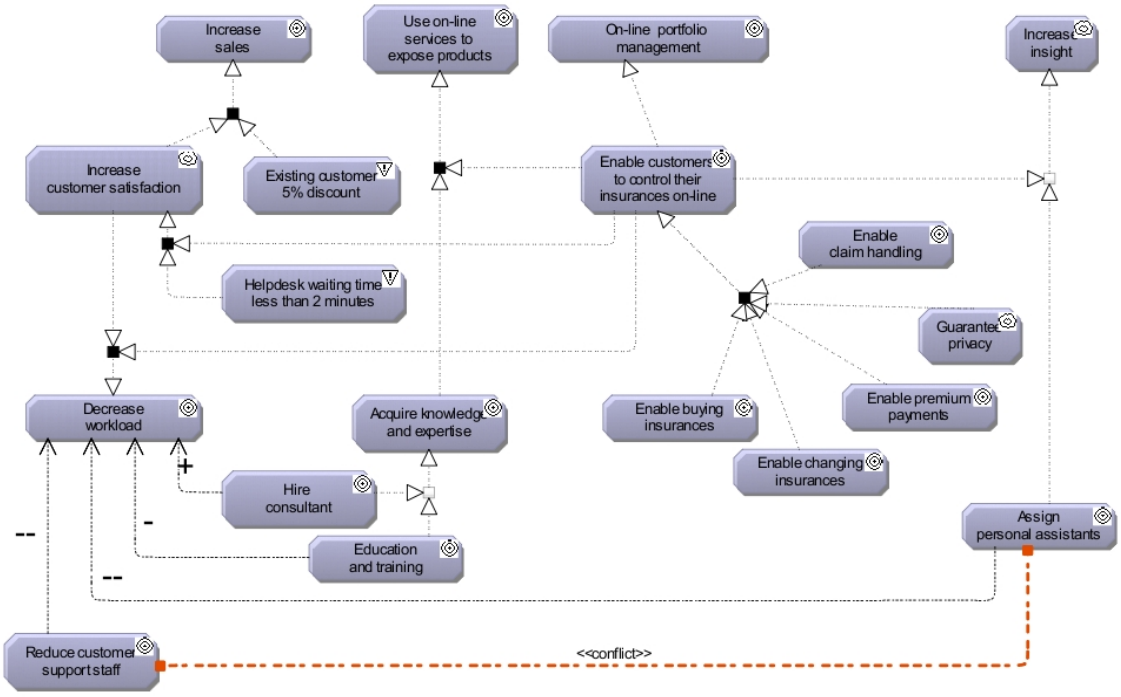
6.9.1.2 Modelleren van high-level goals



Figuur 39: Stakeholders belangen en hun beoordeling d.m.v. goals [QUAR092, p.47]

6.9.1.3 Beschrijving: stakeholders belangen en hun beoordeling d.m.v. goals

Elke beoordeling is toegewezen aan één of meer doelen. In het geval dat een beoordeling een bedreiging of een zwakte betreft, kan een dergelijk doel ontstaan door deze zwakte te onderkennen. De zwakte "Dropped sales" is bijvoorbeeld toegewezen aan het doel "Increase sales". De laatstgenoemde is is weer toegewezen aan een andere bedreiging, namelijk "Leaving customers".



Figuur 40: High-level goal model [QUAR092, p.47]

6.9.1.4 Beschrijving: High-level goal model

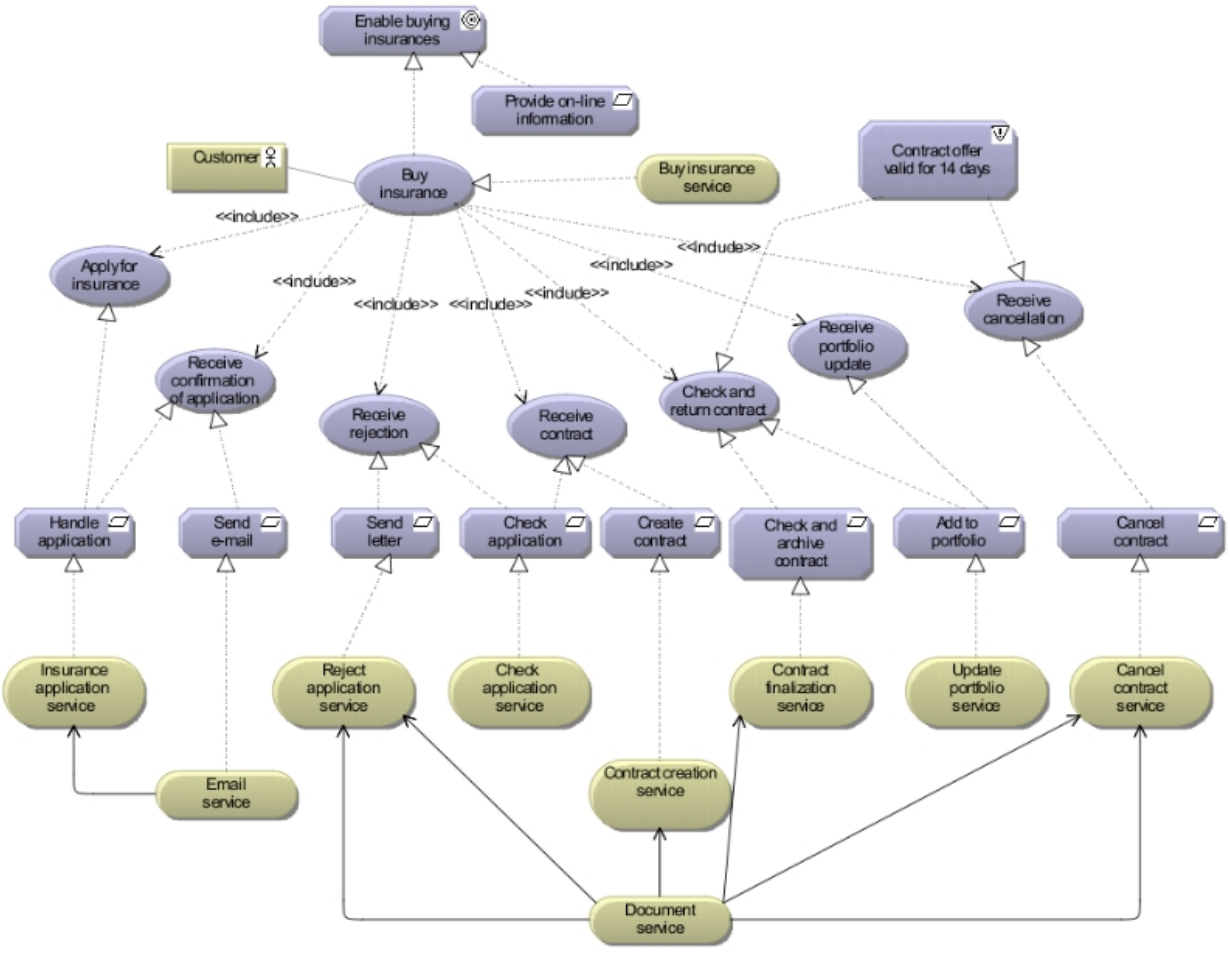
Doelen mogen uitgewerkt worden naar sub-doelen zodat duidelijker gespecificeerd kan worden hoe deze behaalt dienen te worden. Het doel "Increase sales" kan worden behaald door middel van de klanttevredenheid en de ondersteuning voor portfoliomanagement te verbeteren en de bestaande klant 5% korting te geven op de verzekeringspremie. Het laatstgenoemde doel is gemodelleerd als een bedrijfsregel. "Increasing customer satisfaction" is gemodelleerd als een zacht doel (soft goal), omdat er geen criteria is gedefinieerd om te kunnen meten hoe dit doel behaald kan worden. Er wordt direct aangenomen dat het verminderen van de werkdruk en de introductie van online diensten zal bijdragen aan het behalen het doel.

Het doel "Acquire knowledge and expertise" is geïntroduceerd zodat de Service & IT afdeling de online services kunnen ontwikkelen. Dit doel kan behaald worden door twee alternatieve sub-doelen, namelijk het inhuren van een consultant of door opleiding en training. Het eerste sub-doel draagt positief bij aan het doel "Decrease workload", terwijl het tweede doel negatief bijdraagt (het zal namelijk de werkdruk verhogen). Op basis van deze informatie kan men concluderen dat het inhuren van een consultant de

beste optie is.

Het doel "Assign personal assistants (to customers)" en "Reduce customer support staff" zijn gemodelleerd door middel van een conflict doel, aangezien het toewijzen van persoonlijke assistenten aan klanten ervoor zorgt dat de klantondersteuningsstaf wordt vergroot, in plaats van verminderd. Deze twee doelen zijn niet gekoppeld door middel van een conflictrelatie, maar door een negatieve contributie relatie. Dit is bewust gedaan omdat het toewijzen van persoonlijke assistenten niet direct hoeft te leiden voor extra werkdruk. Het verminderen van de werkdruk kan namelijk worden gerealiseerd terwijl de klant alsnog een persoonlijke assistent krijgt toegewezen.

6.9.2 Service requirements



Figuur 41: Van business-goals naar systeem requirements [QUAR092, p.49]

6.9.2.1 Beschrijving: van business-goals naar use-cases systeem requirements

Het doel "Enable buying insurances" wordt gedetailleerder beschreven in de use-case "Buy insurance". De requirements worden geïdentificeerd zodat er online informatie voor de verzekeringsproducten wordt aangeleverd om de klant te ondersteunen tijdens het afnemen van verzekeringen.

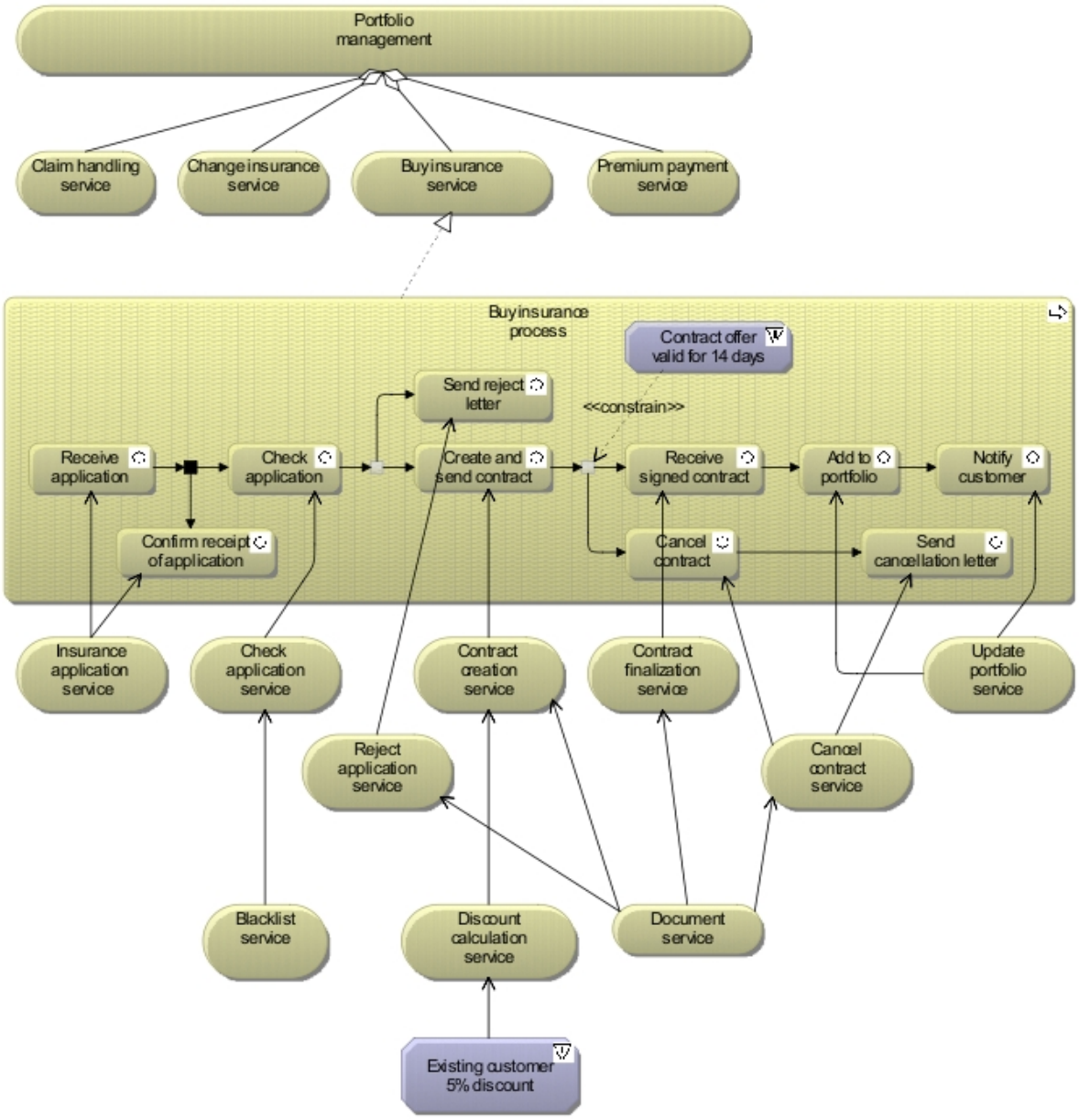
De use-case "Buy Insurance" is uitgewerkt in een aantal kleinere use-cases, welke één of meer (alternatieve) interacties tussen de klant en PRO-Fit representeren. Deze interacties zijn beschreven vanuit het gezichtspunt van de klant. Elke use-case is uitgewerkt naar requirements voor de systemen van PRO-Fit. Deze systeem-requirements ondersteunen deze use-cases. De bedrijfsregel "Contract offer valid for 14 days" is bijvoorbeeld onderdeel van de uitwerkte use-cases "Check and return contract" en "Receive cancellation". Deze bedrijfsregel zorgt ervoor dat een contractaanbod voor een klant maximaal 14 dagen geldig is. Wanneer het contract niet getekend terug wordt verstuurd, wordt het contract geannuleerd. De business service "Buy insurance service" zorgt voor de realisatie van de use-case "Buy insurance". Deze service gebruikt andere services die afgebeeld zijn in het bovenstaande model.

De manier waarop deze services worden gebruikt, wordt in het volgende voorbeeld behandeld.

6.9.2.2 Beschrijving: een businessproces als tegenhanger van een use-case

Dit bedrijfsproces kan gezien worden als tegenhanger van de use-case "Buy insurance". Het beschrijft namelijk het koopproces vanuit het gezichtspunt van PRO-Fit. Het model beschrijft namelijk de activiteiten die PRO-Fit moet uitvoeren en een aantal services die gebruikt worden om deze activiteiten te kunnen realiseren. Bedrijfsregels kunnen namelijk activiteiten of de flow van activiteiten beïnvloeden. De bedrijfsregel "Contract offer valid for 14 days" bepaalt bijvoorbeeld of de activiteit "Create and send contract" moet worden opgevolgd door hetzij activiteit "Receive signed contract" of activiteit "Cancel contract".

Het bedrijfsproces "Discount calculation service" berekent de korting op een verzekering. De bedrijfsregel "Existing customer 5% discount" zorgt ervoor dat er 5% korting wordt toegepast voor bestaande klanten.



Figuur 42: Een businessproces als tegenhanger van een use-case [QUAR092, p.50]

6.10 Mijn bevindingen

Op basis van dit hoofdstuk kan ik de volgende deelvraag beantwoorden:

- **deelvraag 6:** Voldoet ARMOR, de requirementstaal voor enterprise-architectuur aan de gestelde eisen?

6.10.1 Voordelen

ARMOR is een zeer krachtige requirements-taal waarbij de nadruk op motivatie ligt. KAOS, het I*Framework en BPM zijn goal-oriented requirementstalen die zich in de praktijk hebben bewezen en zeer nuttige concepten bevatten. Het is dan ook een groot voordeel dat ARMOR gebaseerd is op deze bestaande talen.

Deze taal hanteert de filosofie dat alle requirements voortkomen uit principes, strategische doelstellingen en belangen van stakeholders. Hierbij worden mijn bevindingen uit het hoofdstuk enterprise-architectuur bevestigd (zie blz. 16). Het is een goal-oriented requirementstaal waarbij het concept “doel” centraal staat. Het onderscheid tussen harde en zachte doelen maakt het mogelijk om een hard doel uit te werken naar meerdere sub-doelen. Dit gebeurt door middel van een doelverfijning relatie. Overige relaties zijn de conflict- en alternatiefrelatie. Met deze relaties is het mogelijk om een conflict tussen twee doelen te modelleren en de keuze voor een eventueel alternatief. Een requirement is een type doel en een verwachting is een type requirement. Het is mogelijk om een doel op hoog niveau te herleiden naar een concrete requirement.

Het stakeholders-domein modelleert de stakeholders per architecturale laag met hun belangen en bijbehorende beoordelingen. Het principles-domein modelleert de visie, missie, strategie, policy, principes en richtlijnen. Deze twee laatstgenoemde domeinen zorgen voor de input van het requirements-domein. Dit domein modelleert de doelen, requirements en verwachtingen die tevens het ontwerp van de enterprise inperken. Het rationale-domein is een optionele uitbreiding waarmee ontwerpkeuzes gemodelleerd kunnen worden. Overige optionele domeinen zijn het business rules en use-cases domein. Een bedrijfsregel en een use-case worden beschouwd als een type requirement. Een bedrijfsregel is een regel die een beperking definieert voor de business van een enterprise. Use-cases worden voornamelijk gebruikt voor het ontlokken en specificeren van requirements. Deze twee domeinen zijn de enige domeinen die betrekking hebben op het operationele gedrag van een enterprise.

De koppeling met ArchiMate is intelligent en zorgt ervoor dat doelen gekoppeld kunnen worden met concepten uit een ea-model. Op deze manier is het mogelijk om elementen uit het principles- of

stakeholders-domein te herleiden naar requirements en vervolgens de requirements te herleiden naar artefacten uit het ea-model die verantwoordelijk zijn voor de realisatie van deze requirements.

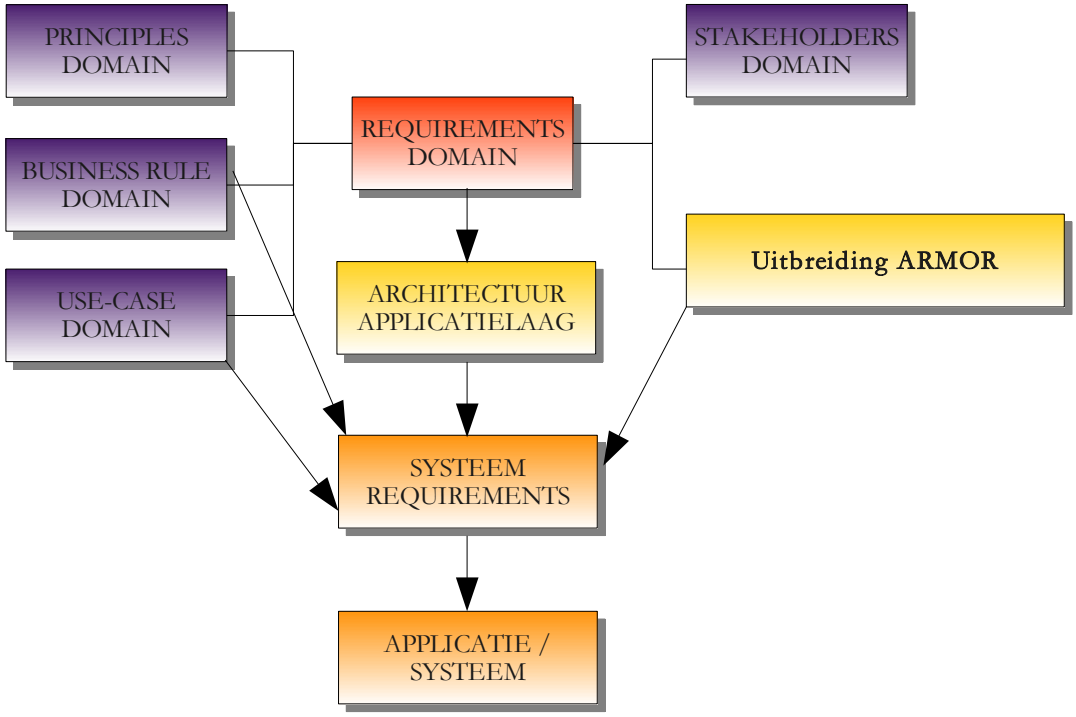
6.10.2 Nadelen: Van architectuurniveau naar systeemniveau

ARMOR is een intelligente requirementstaal voor enterprise-architectuur. Wat echter een nadeel is van de taal, is dat het modelleren zich voornamelijk beperkt in de hoge laag van de enterprise waarbij het requirements-traceability-proces 1 (zie blz. 34) uitgevoerd kan worden. Wat uit het hoofdstuk enterprise-architectuur is af te leiden, heeft een architectuur invloed op het operationele niveau en bepaalt de architectuur de functionaliteit van deze systemen. Ook wanneer er sprake is van een systeembeperking, kan dit systeem op operationeel niveau de architectuur beïnvloeden.

ARMOR heeft de mogelijkheid om use-cases en bedrijfsregels te modelleren, wat echter maar een klein deel van het operationele niveau is. Het requirements-traceability-proces 2 (zie blz. 35) wordt matig ondersteund in ARMOR. In ArchiMate is het mogelijk om applicatiemodellen te koppelen met concepten uit een ea-model, zodat het mogelijk is om verder in te zoomen op de functionaliteit van een applicatie. Het lijkt mij niet alleen belangrijk om de requirements van deze systemen te kunnen modelleren maar ook inzicht te verkrijgen in de functionaliteit van deze systemen.

7 Verantwoording uitbreiding voor ARMOR

In mijn bevindingen van het vorige hoofdstuk licht ik toe dat het uitvoeren van traceability tussen architectuur en een systeem zeer beperkt is. Ik wil een uitbreiding toevoegen waarmee mogelijk wordt gemaakt om het traceability proces 2 (zie blz. 35) uit te voeren. Ik probeer in het onderstaande figuur de huidige situatie in kaart te brengen:



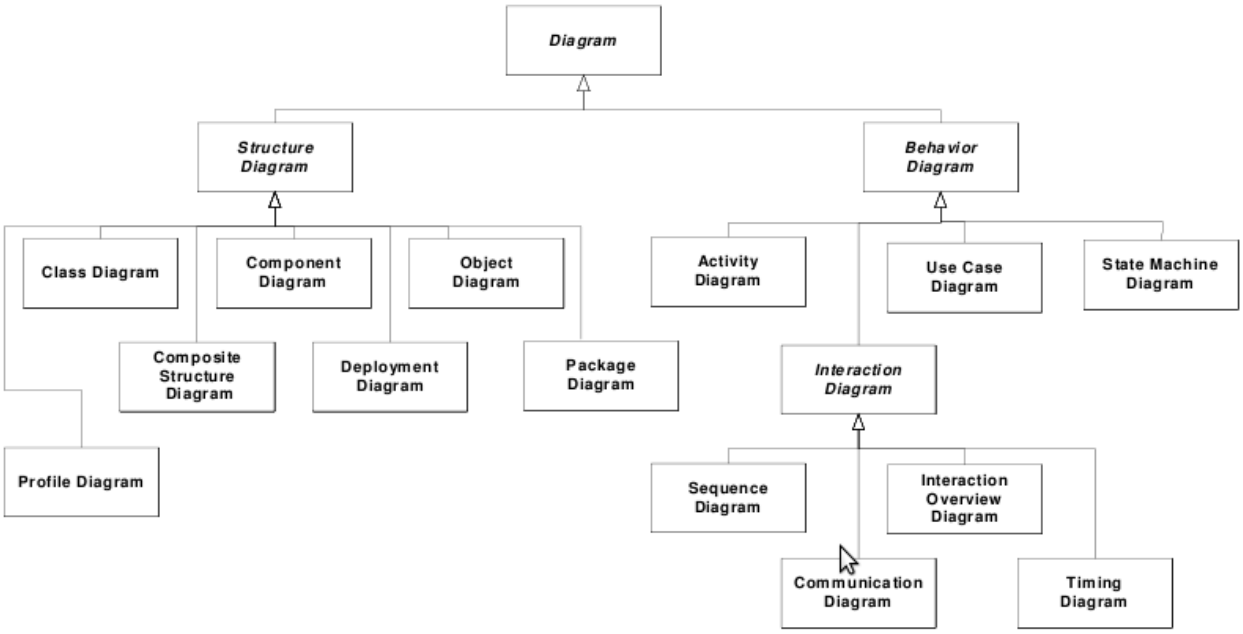
Figuur 43: Uitbreiding ARMOR

De paarse vakken zijn van ARMOR, "architectuur applicatielaag" is onderdeel van ArchiMate, de oranje vakken zijn ter verduidelijking.

Gedetailleerde beschrijvingen van artefacten uit een applicatielaag ea-model, worden doorgaans gemodelleerd in UML (Unified Modelling Language). UML is een standaard voor het modelleren en ontwerpen van systemen en applicaties. ARMOR, maar ook ArchiMate zijn bijvoorbeeld ontworpen met deze modelleertaal. In de volgende paragraaf ga ik UML onderzoeken en onderbouwen welk type model interessant is voor de uitbreiding van ARMOR.

7.1 Unified Modelling Language

UML is een modelleertaal om objectgeoriënteerde analyses en ontwerpen voor een informatiesysteem/applicatie te kunnen maken. De taal is in de begin jaren '90 ontworpen door Booch, Rumbaugh en Jacobson en is in 1997 een standaard geworden en wordt nu beheerd door de Object Management Group (OMG). UML biedt een verzameling van statische en dynamische diagrammen. Elk diagram beschrijft een bepaald aspect van een systeem. Deze diagrammen zijn onderverdeeld in drie verschillende categorieën, namelijk structuur, gedrag en interactie.



Figuur 44: Beschikbare UML diagrammen [BOOC99]

7.1.1 Structuur

Class diagram

Een klassendiagram beschrijft de structuur van het systeem (op typeniveau), door het gebruik van klassen en relaties tussen deze klassen. Een klassendiagram beschrijft de type objecten die een systeem kent en laat zien welke statische relaties, waarvan er verschillende kunnen zijn, tussen deze objecten bestaan. Een klasse representeert de gezamenlijke kenmerken en het gemeenschappelijke gedrag van een verzameling objecten.

Component diagram

Een componentdiagram modelleert de afhankelijkheden tussen componenten. Daarbij ligt de nadruk

op zowel de interfaces die een component leveren, als op de interfaces welke een component nodig heeft om te kunnen functioneren. Een component realiseert één of meerdere interfaces. De realisatie van deze interfaces is door het component ingekapseld.

Object diagram

Een objectdiagram beschrijft de objecten binnen een applicatie of de structuur van het systeem (op instantieniveau) op een bepaald moment, door het gebruik van objecten en relaties tussen deze objecten. Objecten zijn instanties van klassen en hebben een unieke identiteit. De attributen van een object hebben een bepaalde waarde op een bepaald tijdstip.

Composite diagram

Dit diagram maakt het mogelijk om de interne structuur van een klasse of een component te modelleren. Het diagram toont de samenwerkende onderdelen waaruit de structuur is opgebouwd.

Deployment diagram

Het deployment diagram beschrijft welke middelen benodigd zijn om de componenten hun werk te laten doen. Voor applicaties zijn dit bijvoorbeeld middelen zoals hardware; geheugen, processor etc.

Package diagram

Dit diagram geeft weer hoe bepaalde modelelementen zijn gegroepeerd in een pakket en de afhankelijkheden tussen bepaalde pakketten. Een pakket kan individuele elementen of andere pakketten importeren.

7.1.2 Gedrag

Activity diagram

Een activiteitendiagram beschrijft de toestanden van het systeem en hoe de verschillende toestanden in elkaar overlopen. Het beschrijft de werkstroom (workflow) van een softwarecomponent.

Use case diagram

Een use-case is een beschrijving van het gedrag van een systeem, dat reageert op een verzoek dat stamt van buiten het systeem (actor). Samengevat beschrijft een use-case "wie" met het betreffende systeem "wat" kan doen. Use-cases worden doorgaans gebruikt voor het achterhalen en specificeren van de requirements, waarbij de interactie tussen het systeem en de gebruiker centraal staat.

State machine diagram

Dit diagram geeft de levenscyclus van een individuele klasse of een klein deel uit de applicatie weer. Deze techniek heeft alleen zin als deze cyclus diverse statussen en statusovergangen kent.

7.1.3 Interactie

Communication diagram (voorheen collaboration)

Het communicatiediagram toont hoe het systeem gebruikt gaat worden en welke handelingen ermee worden verricht. Er wordt getoond welke verantwoordelijkheden (berichten / methoden) worden verstuurd over de associatie.

Sequence diagram

Het sequence diagram geeft de interacties weer tussen verschillende objecten (instanties van de klasses) welke een bepaalde functionaliteit (of een deel ervan) implementeren en diens berichtenstroom. De tijdsvolgorde staat centraal in dit diagram. In tegenstelling tot het communication diagram, is een dergelijke berichtenstroom niet beperkt tot één maar tot meerdere scenario's.

Interaction overview diagram

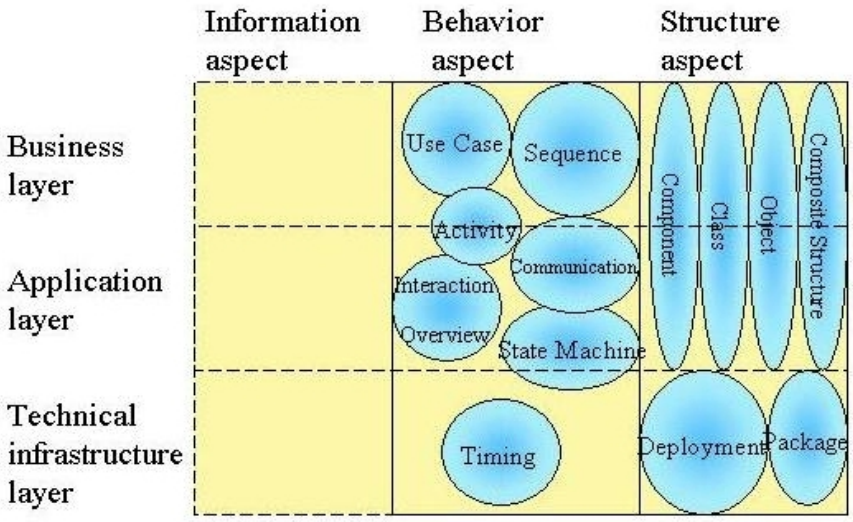
Dit diagram combineert de flowstructuur van een activity diagram met een sequence diagram. Het vervangt de activiteiten met sequence diagrammen. Op deze manier kan de interactie op een hoog niveau worden gemodelleerd.

Timing diagram

Een timing diagram combineert een sequence diagram en een expliciete tijd. Het beschrijft voornamelijk de veranderingen in de status van één individueel object over tijd.

7.2 De link tussen UML en ArchiMate

De verschillende UML modellen zijn niet te groeperen in business, applicatie en technologie, omdat UML oorspronkelijk ontwikkeld is voor het ontwerpen van softwaresystemen. Er kan echter wel onderscheid gemaakt worden tussen de statische en dynamische modellen. De stippellijnen in het onderstaande figuur geven aan dat het geen officiële grens betreft.



Figuur 45: UML Viewpoints in ArchiMate Framework [IACO04]

7.3 Verantwoording keuze model

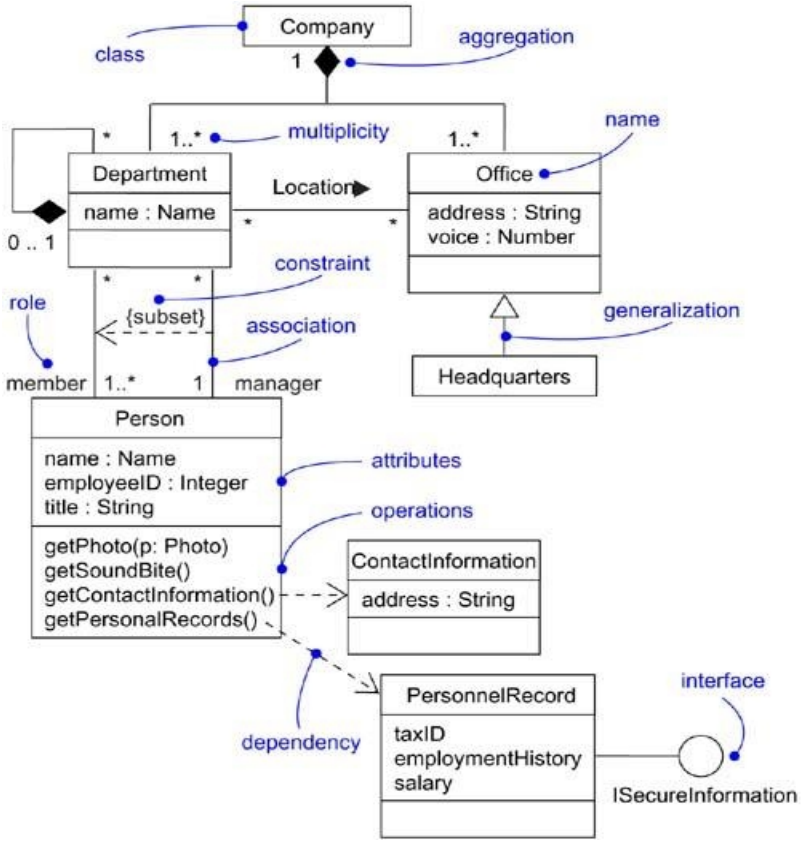
UML biedt een breed scala van modellen aan die een belangrijke rol spelen bij de ontwikkeling van een softwaresysteem. Ik kies voor het klassendiagram omdat dit model een goed beeld geeft van de algemene functionaliteit van het systeem. De overige modellen zoomen in op een specifiek aspect van het systeem of visualiseren de communicatie tussen verschillende objecten en klassen. ARMOR biedt reeds de mogelijkheid om use-cases te modelleren. Deze use-cases worden gebruikt voor de ontwikkeling van een klassendiagram, aangezien het doel of de requirement van een use-case gerealiseerd wordt door één of meerdere klassen uit het klassendiagram.

7.4 Klassendiagram

7.4.1 Inleiding

Klassendiagrammen zijn de meest voorkomende diagrammen binnen objectgeoriënteerd modelleren. Een klassendiagram geeft een verzameling klassen, interfaces en samenwerkingen (collaborations) met hun relaties weer. Een klassendiagram wordt gebruikt om het statische ontwerp van een systeem te visualiseren. Meestal wordt een dergelijk diagram gebruikt om de vocabulaire van een systeem, de samenwerkingen en (database) schema's te modelleren. Verder is een klassendiagram de basis voor twee andere UML diagrammen, namelijk het component diagram en deployment diagram.

Samengevat bevat een klassendiagram uit de volgende onderdelen: klassen, interfaces, samenwerkingen en relaties.

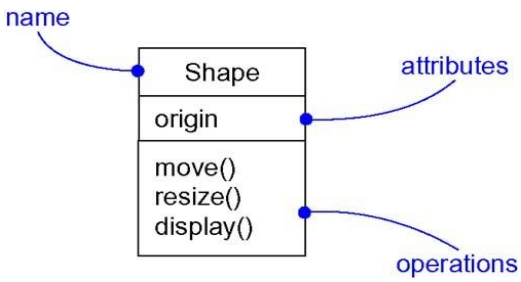


Figuur 46: Klassendiagram (UML) [BOOC99]

7.4.2 Klassen

Klassen zijn zeer belangrijk bij het ontwikkelen van een object-georiënteerd systeem. Een klasse is een beschrijving van een aantal objecten die dezelfde attributen, operaties, relaties en semantiek delen. Een klasse implementeert één of meerdere interfaces. Een klassendiagram wordt gebruikt om het statische ontwerp van een systeem weer te geven. Klassen worden gebruikt om het vocabulaire van het systeem vast te leggen. Deze klassen bevatten zowel abstracties die deel zijn van het probleemdomein, als klassen die zorgen voor de implementatie. Klassen kunnen gebruikt worden om software of hardware te modelleren, maar ook voor conceptuele ontwerpen. Gestructureerde klassen hebben duidelijke grenzen en zorgen voor een gebalanceerde distributie en verantwoordelijkheid van een systeem.

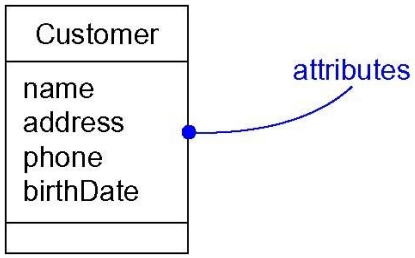
Een klasse wordt getekend door middel van een vierkant. Een klasse bevat een naam, attributen, operaties en verantwoordelijkheden.



Figuur 47: Voorbeeld klasse (UML) [BOOC99]

Attributen

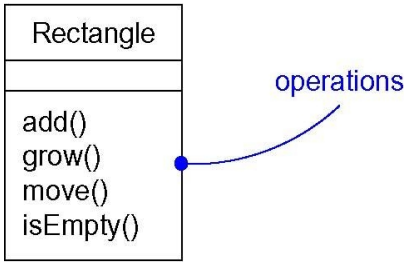
Een attribuut beschrijft een eigenschap van een klasse, en niet van een individuele instantie. Een klasse kan een onbeperkt aantal attributen bevatten of helemaal geen. Een attribuut geldt voor alle objecten van een klasse. Hieronder geef ik een voorbeeld van de klasse "klant". Voor elke klant geldt dat deze een naam, adres, telefoonnr en geboortejaar heeft.



Figuur 48: Attributen (UML) [BOOC99]

Operaties

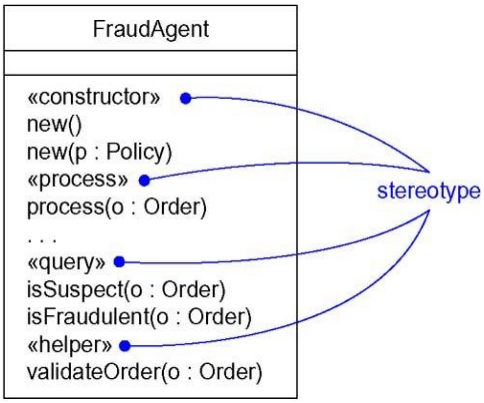
Een operatie is de implementatie van een service die het systeem kan uitvoeren. Deze service wordt aangeroepen door een object van de desbetreffende klasse. Een operatie is een abstractie van iets wat men kan doen met een object en wat gedeeld wordt door alle objecten in deze klasse. Een klasse kan een onbeperkt aantal of helemaal geen operaties bevatten. Hieronder een voorbeeld van de klasse "vierkant" welke de volgende operaties bevat: toevoegen, vergroten, verplaatsen en leegmaken.



Figuur 49: Operations (UML) [BOOC99]

Organiseren van attributen en operaties

Het is niet verplicht om alle operaties en attributen in de klasse te modelleren. Het aantal kan bij sommige projecten flink oplopen en het weergeven van alleen de relevante eigenschappen is dan meestal genoeg. Door middel van "..." kan men aangeven dat er nog meer eigenschappen bij de klasse toebehoren. Het is ook mogelijk om de operaties en attributen onder categorieën onder te brengen.

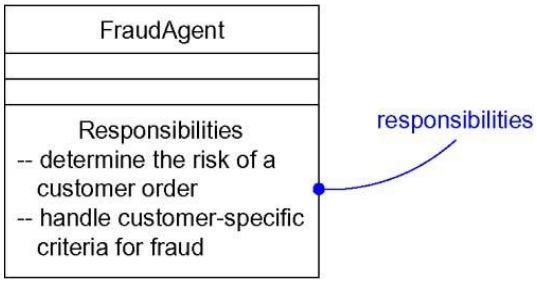


Figuur 50: Organiseren van operaties en attributen (UML) [BOOC99]

Verantwoordelijkheden

Een verantwoordelijkheid is een contract of een obligatie van een klasse. Wanneer een klasse wordt gemaakt, gaat men ervan uit dat alle objecten binnen deze klasse hetzelfde gedrag hanteren. De attributen en operaties ondersteunen de verantwoordelijkheid van een klasse. De klasse "muur" is verantwoordelijk voor het weten van de breedte, hoogte en dikte, de klasse "FraudAgent" (klasse van een creditcardapplicatie) is verantwoordelijk voor het controleren van de orders, of deze wel legitiem, niet verdacht of gefraudeerd zijn.

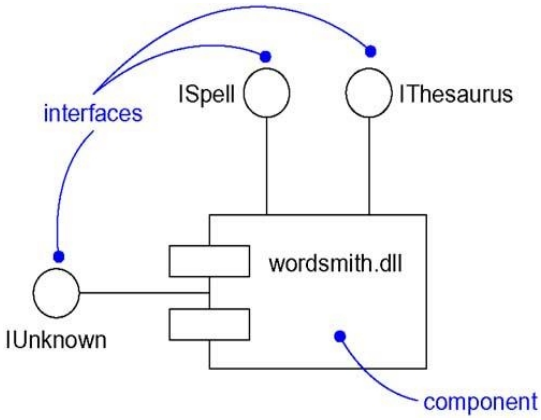
Bij het modelleren van klassen is het wijsheid om eerst de verantwoordelijkheden te benoemen. Use-cases en andere requirementstechnieken worden hiervoor gebruikt. Een goed ontworpen klasse moet minimaal één verantwoordelijkheid hebben. Dermate het ontwikkelingsproces worden de verantwoordelijkheden vertaald naar attributen en operaties.



Figuur 51: Verantwoordelijkheden klasse (UML) [BOOC99]

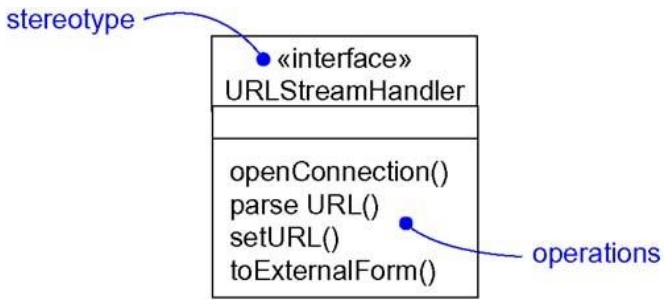
7.4.3 Interfaces

Een interface is een verzameling operaties die een service van een klasse of component specificeren. Interfaces worden gebruikt om de grens binnen het systeem te visualiseren, te specificeren, te bouwen en te documenteren. Een gestructureerde interface zorgt voor een duidelijke scheidingslijn tussen het gezichtspunt van de buitenkant en het gezichtspunt van de binnenkant van een abstractie.



Figuur 52: Interfaces (UML) [BOOC99]

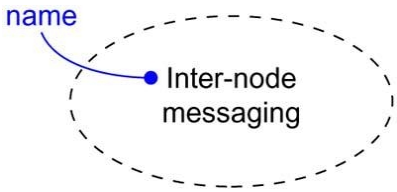
In tegenstelling tot een klasse, is er geen structuur voor een interface gespecificeerd en mogen interfaces geen attributen of methoden (voor de implementatie van een operatie) bevatten. Een interface wordt doorgaans met een cirkel gemodelleerd, het is echter ook mogelijk om een interface als een klasse te modelleren zodat de betekenis duidelijker is.



Figuur 53: Operations Interface (UML) [BOOC99]

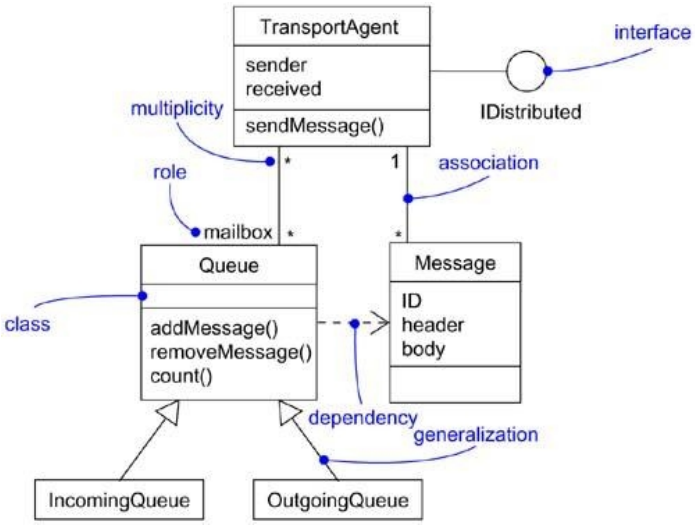
7.4.4 Samenwerking

Een samenwerking is een samenleving van klassen, interfaces en andere elementen die samen kunnen werken en zorgen voor een bepaald gedrag. Een samenwerking is ook een specificatie van hoe een element zoals een classifier (klasse, interface, component etc.) of een operatie wordt gerealiseerd.



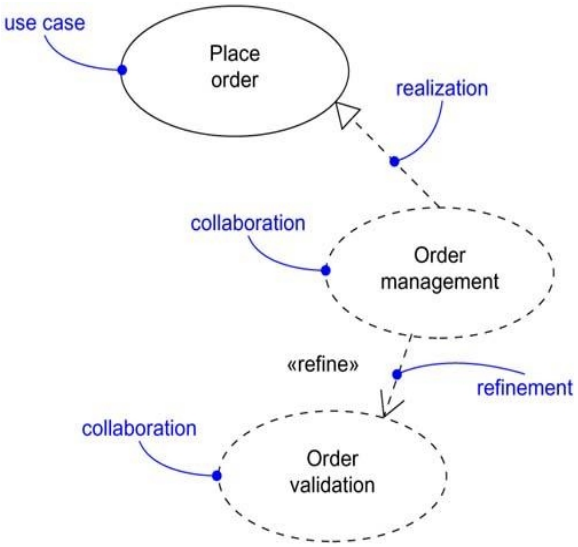
Figuur 54: Collaboration (UML) [BOOC99]

Een samenwerking bestaat uit twee aspecten, namelijk het structurele aspect en het gedrag van de samenwerking. Het structurele aspect beschrijft de klassen en interfaces van de samenwerking.



Figuur 55: Structural Aspects of a Collaboration (UML) [BOOC99]

Het gedrag beschrijft de interactie tussen de klassen en interfaces.



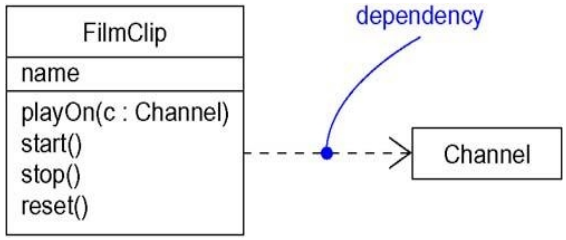
Figuur 56: Behaviour Aspect of a Collaboration [BOOC99]

7.4.5 Relaties

Om klassen, interfaces en samenwerkingen te kunnen koppelen, maakt men gebruik van relaties. De meest voorkomende relaties binnen objectgeoriënteerd modelleren zijn: afhankelijkheden (dependencies), generalisaties (generalizations) en associaties (associations).

7.4.5.1 Afhankelijkheidsrelatie

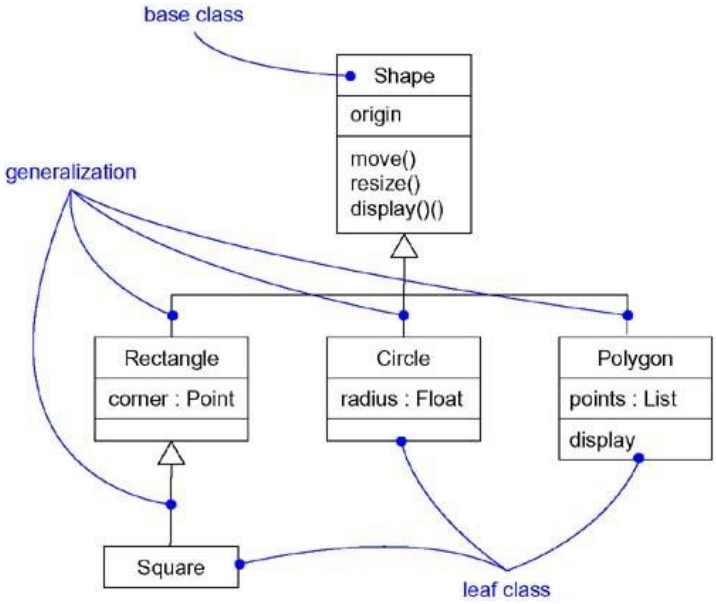
Een afhankelijkheidsrelatie wordt gebruikt om aan te geven dat een bepaald element (bijvoorbeeld Steen) afhankelijk is van een ander element (bijvoorbeeld Muur), maar niet expliciet andersom. Dit type relatie wordt voornamelijk gebruikt in de context van klassen; wanneer een bepaalde klasse een andere klasse nodig heeft als argument om een specifieke operatie uit te voeren.



Figuur 57: Dependency relation (UML) [BOOC99]

7.4.5.2 Generalisatierelatie

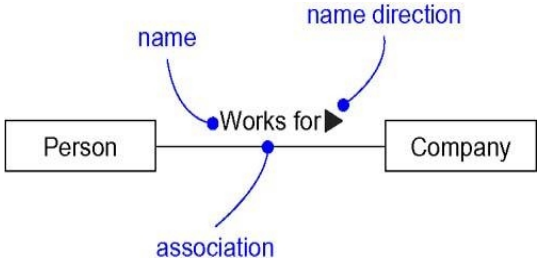
Een generalisatie is een relatie tussen een algemeen element (superklasse of ouder) en een meer specifiek element (subklasse of kind). Dit type relatie wordt ook wel "is-een-kind-van" relatie genoemd: een element (gipsMuur) is een kind van een algemener element (Muur). Generalisatie betekent dat objecten van het kind gebruikt kunnen worden waar de ouder voorkomt, maar niet andersom. Het kind overerft de operaties en attributen van de ouder.



Figuur 58: Generalization relation (UML) [BOOC99]

7.4.5.3 Associatierelatie

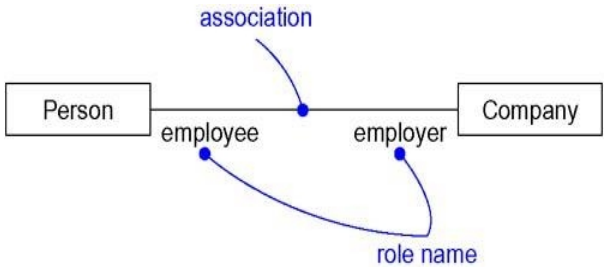
Een associatie specificeert een structurele relatie tussen objecten van een bepaald element met objecten van een ander element. Wanneer twee klassen met een associatierelatie met elkaar zijn gekoppeld, is het mogelijk om van een object uit de ene klasse te navigeren naar een object uit de andere klasse, en vice versa.



Figuur 59: Name Assocation Relation (UML) [BOOC99]

Rol

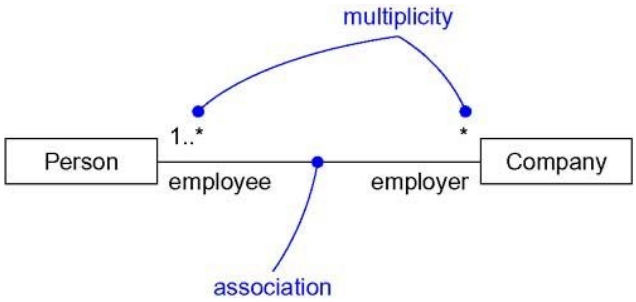
Wanneer een klasse is gekoppeld met een associatierelatie, bekleed deze klasse een rol. Deze rol wordt gebruikt op het einde van de relatie die gepresenteerd wordt aan de andere klasse.



Figuur 60: Rol Assocation Relation (UML) [BOOC99]

Veelheid (multiplicity)

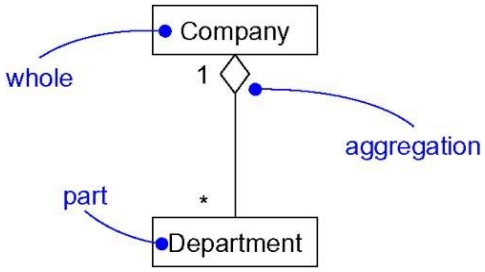
Een associatie representeert een structurele relatie tussen objecten. Het is belangrijk om aan te geven hoeveel objecten onderdeel zijn van een associatie. Dit wordt gedaan door de hoeveelheid te modelleren.



Figuur 61: Multiplicity (UML) [BOOC99]

Aggregatierelatie (aggregation)

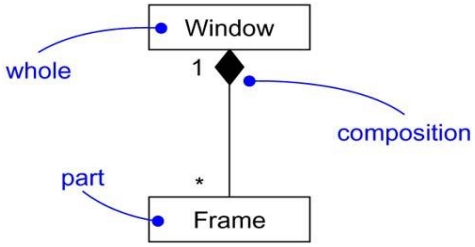
Soms is er behoefte om een relatie te modelleren tussen een geheel en een onderdeel. Een klasse die het geheel representeert en bestaat uit verschillende onderdelen. Dit wordt een aggregatierelatie genoemd.



Figuur 62: Aggregation (UML) [BOOC99]

Compositie

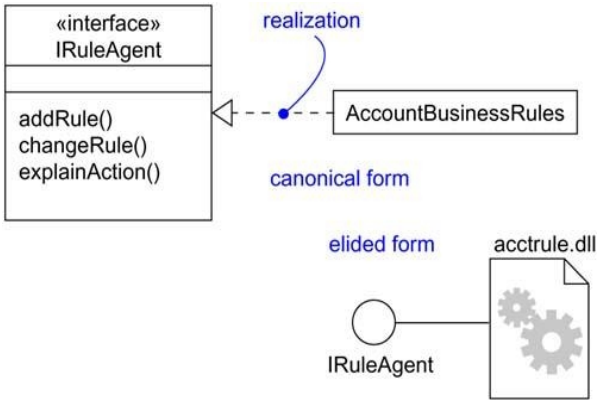
Compositie is een variant op de aggregatierelatie. Deze relatie voegt sterke eigendom- en levenscycluseigenschappen toe.



Figuur 63: Composition (UML) [BOOC99]

7.4.5.4 Realisatierelatie

Een realisatierelatie wordt meestal gebruikt tussen een klasse en een interface. De entiteit interface definieert een verzameling functionaliteiten voor de entiteit klasse. Een realisatierelatie is eigenlijk een mix tussen een afhankelijkheidsrelatie en een generalisatierelatie.



Figuur 64: Realization (UML) [BOOC99]

7.5 Requirements van het klassendiagram

De ontwikkeling van een klassendiagram is gebaseerd op een verzameling requirements. Deze requirements zijn gebaseerd op de volgende bronnen:

7.5.1 Enterprise-architectuur

Mijn bevindingen uit het hoofdstuk enterprise-architectuur wijzen uit dat enterprise-architectuur invloed uitoefent op het operationele niveau van een organisatie. Enterprise-architectuur beschrijft hoe de organisatie, de informatievoorziening, de applicaties en de infrastructuur hun vorm hebben gekregen en hoe zij zich voordoen in het gebruik. De inrichting van de enterprise-architectuur vormen de basis van de requirements voor een klassendiagram.

7.5.2 Use-cases

Use-cases zijn een middel om requirements boven water te krijgen, het beschrijft "wat" een systeem moet kunnen. Een klassendiagram beschrijft "hoe" een systeem werkt. Het maken van een klassendiagram gebeurt door voor elke requirement, "wat" te converteren naar "hoe". Een use-case wordt geanalyseerd en in atomaire componenten opgedeeld, welke de basis vormt voor de klassen die ontwikkeld moeten worden.

7.5.3 Overige artefacten

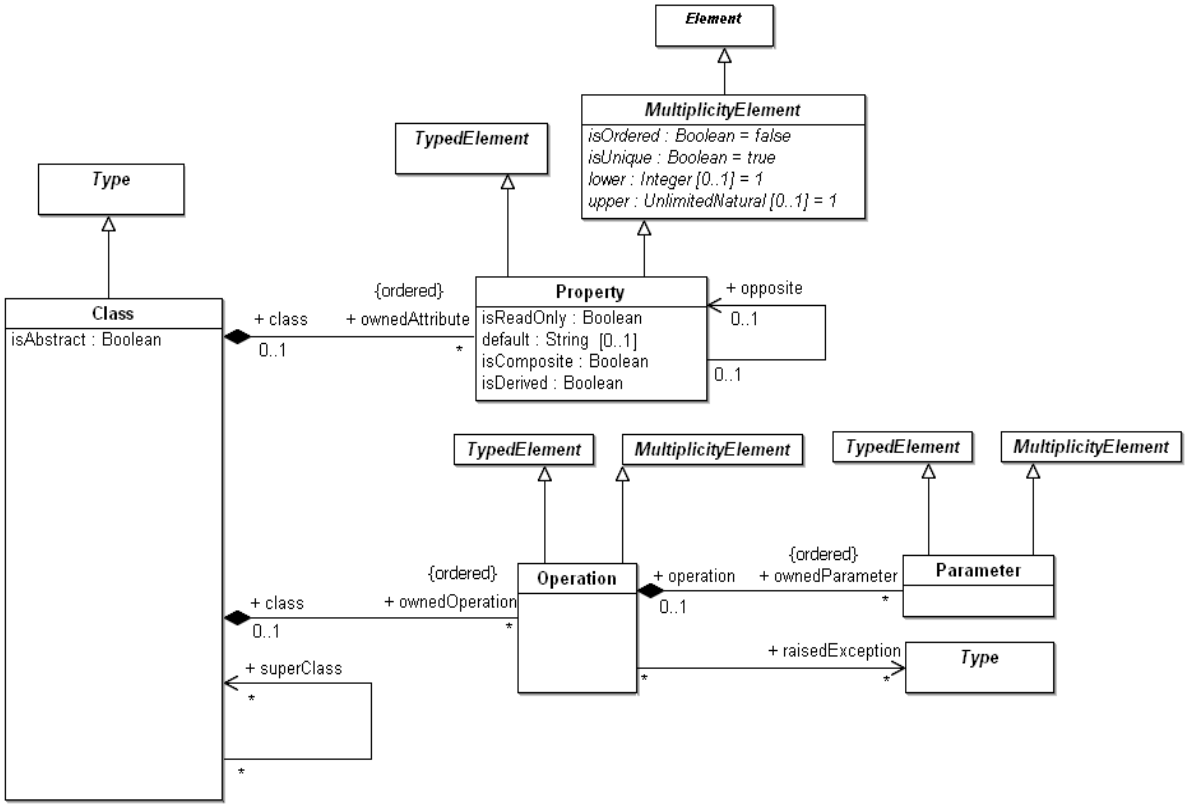
Alhoewel enterprise-architectuur en use-cases een groot deel van de lading dekken, kunnen er nog andere artefacten zijn die de requirements bepalen. Denk hierbij aan projectdocumenten, functionele specificaties of belangen van stakeholders die zich op een lager niveau binnen de organisatie bevinden. Deze overige artefacten abstraheer ik in dit onderzoek. Ik zal mij volledig richten op de input van enterprise-architectuur en de use-cases.

8 Integratie UML klassendiagram en ARMOR

8.1 Meta-model van het UML klassendiagram

UML is gebaseerd op een architectuur van vier meta-modellering niveaus. Elk niveau is gelabeld van M^3 tot M^0 en worden meestal verwoord als het meta-metamodel, metamodel, klassendiagram en objectdiagram. Een diagram op het niveau M is een instantie van het diagram op niveau $Mi+^1$. Hiervoor geldt dat een objectdiagram (M^0-niveau diagram) een instantie is van een klassendiagram (M^1-niveau diagram) en dat dit klassendiagram een instantie is van het meta-model (M^2-niveau diagram). Het M^3-niveau diagram wordt gebruikt om de structuur van een meta-model te definiëren, waarbij ook het Meta Object Facility (MOF) behoord. Het UML meta-model bevindt zich op het M^2-niveau.

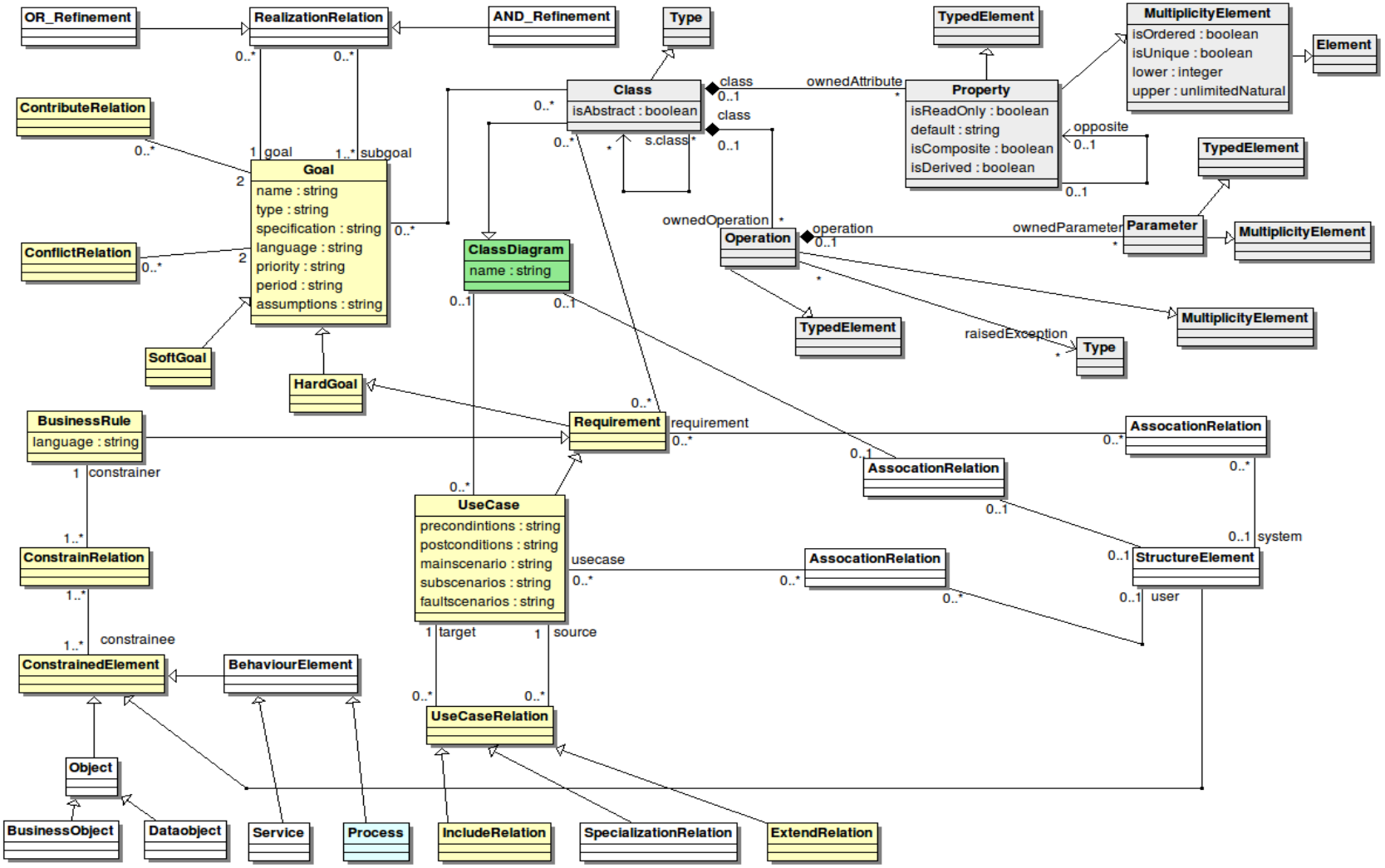
Hieronder volgt het meta-model van het klassendiagram zoals deze is gedefinieerd in [OMG09, p.95].



Figuur 65: Meta-model van het klassendiagram [OMG09, p. 95]

De volgende stap is om dit meta-model te implementeren en te koppelen met het ARMOR meta-model (zie blz. 55).

8.2 Uitgebreid meta-model ARMOR



Figuur 66: Uitgebreid meta-model ARMOR

8.3 Beschrijving

De gele en de witte klassen zijn onderdeel van het ARMOR meta-model. Er is onderscheid gemaakt tussen geel en wit omdat de witte klassen uit het ArchiMate meta-model afkomstig zijn. De aanwezigheid van deze ArchiMate concepten zorgt ervoor dat ARMOR binnen een ArchiMate-model gebruikt kan worden. Groene klassen zijn nieuwe concepten die ik aan ARMOR heb toegevoegd en blauwe klassen zijn concepten van het UML klassendiagram meta-model.

De implementatie van het UML meta-model heb ik gerealiseerd door de klassen ClassDiagram en Class te koppelen met concepten uit ARMOR. De overige klassen uit het UML klassendiagram spelen geen rol bij de implementatie en ondersteunen alleen de klasse "Class" en zijn verantwoordelijk voor de syntax en semantiek van het klassendiagram.

Na het bestuderen van het ARMOR meta-model heb ik de conclusie getrokken dat het noodzakelijk is om gebruik te maken van het ArchiMate-concept "AssociationRelation" wanneer er een relatie wordt gelegd tussen een concept van ARMOR en een concept van ArchiMate. Ik kies er bewust voor om het "AssociationRelation" concept meerdere malen in het model te gebruiken.

8.3.1 Relatie tussen ClassDiagram en Class

Ik heb een nieuw ARMOR-concept geïntroduceerd, namelijk ClassDiagram. Dit concept representeert een voltooid klassendiagram op basis van het UML klassendiagram meta-model. Deze koppeling zorgt ervoor dat er eenvoudig relaties tussen het klassendiagram en overige concepten kunnen worden gelegd.

8.3.2 Relatie tussen ClassDiagram en StructureElement

Een klassendiagram is altijd gelinkt met één systeem en één systeem kan altijd maar met één klassendiagram gelinkt zijn. De aanwezigheid van deze relatie biedt de mogelijkheid om een klassendiagram te koppelen aan een applicatie of een applicatiecomponent, zodat het mogelijk is om verder in te zoomen op de functionaliteit van het systeem. Omdat ClassDiagram gekoppeld wordt met een ArchiMate concept, maak ik gebruik van het "AssociationRelation" concept.

8.3.3 Relatie tussen ClassDiagram en UseCase

Deze relatie heb ik gerealiseerd omdat een use-case de ontwikkeling en het opstellen van een klassendiagram ondersteund. In het model kan men meteen het doel van een use-case herleiden in het klassendiagram. Voor één klassendiagram geldt dat er nul of meerdere use-cases zijn gekoppeld. Voor

één use-case geldt dat deze altijd maar met één systeem gekoppeld mag zijn.

8.3.4 Relatie tussen Class en Requirement

Door deze relatie kan men een (functionele) requirement koppelen met de klasse die verantwoordelijk is voor deze requirement. Omdat een klasse de operaties weergeeft kan men zien welke operaties ervoor zorgen dat de requirement behaald wordt. Hiervoor geldt dat er één of meerdere klassen verantwoordelijk kunnen zijn voor één requirement. Meerdere requirements kunnen aan één klasse gekoppeld zijn.

8.3.5 Relatie tussen Goal en Class

Ik heb er voor gekozen om de klasse "Goal" te koppelen met de klasse "Class" zodat het mogelijk is om de operationele invloed van een doel te herleiden naar de functionaliteit van een systeem. Deze relatie zal niet worden gebruikt wanneer er op een prescriptieve manier enterprise-architectuur wordt bedreven aangezien het verfijnen van de doelen hiervoor noodzakelijk is. Deze relatie kan worden gebruikt wanneer men al in kaart heeft gebracht welke klassen verantwoordelijk zijn voor specifieke doelen en men de behoefte heeft om een model te maken waarbij deze onderlinge relaties direct zichtbaar zijn.

8.4 Relatie matrix

Concept	Concept	Relatie	Natuurlijke taal
ClassDiagram (ARMOR)	Class (UML)	Generalisatie	De klasse ClassDiagram overerft de eigenschappen van het UML meta-model.
ClassDiagram (ARMOR)	UseCase (ARMOR)	Associatie	Een use-case kan maximaal één klassendiagram ondersteunen, maar een klassendiagram kan ondersteund worden door meerdere use-cases.
ClassDiagram (ARMOR)	AssociationRelation (ArchiMate)	Associatie	Een klassendiagram kan aan nul of maximaal één AssociationRelation gekoppeld zijn. Een AssociationRelation kan aan nul of maximaal één klassendiagram gekoppeld zijn.
AssociationRelation (ArchiMate)	StructureElement (ArchiMate)	Associatie	Een AssociationRelation kan aan nul of maximaal één StructureElement gekoppeld zijn. Een StructureElement kan aan maximaal één AssociationRelation gekoppeld zijn.
Class (UML)	Requirement (ARMOR)	Associatie	Een klasse is verantwoordelijk voor nul of meerdere requirements en een requirement kan worden behaald door nul of meerdere klassen.
Class (UML)	Goal (ARMOR)	Associatie	Een goal kan door nul of meerdere klassen ondersteund worden. Een klasse kan nul of meerdere goals ondersteunen.

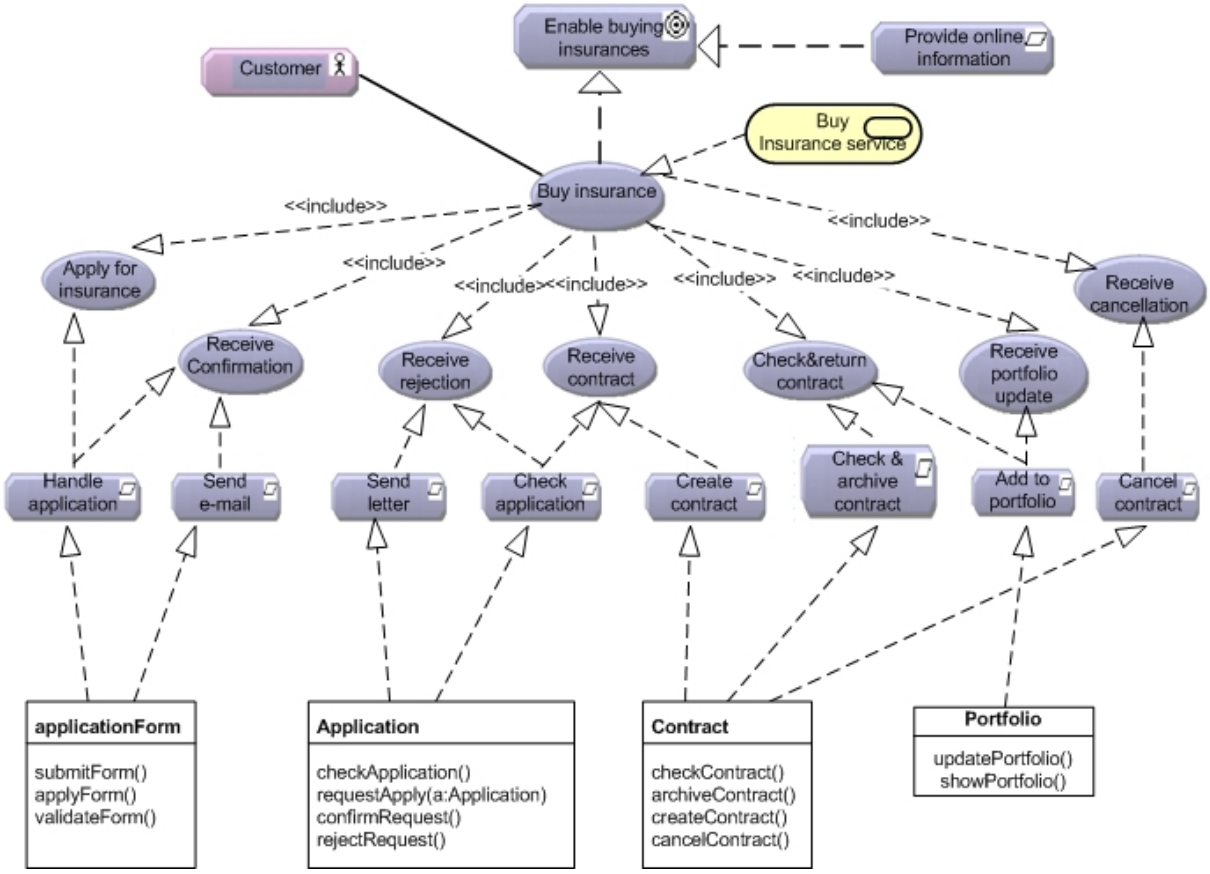
Tabel 8: Nieuwe relaties van de ARMOR uitbreiding

8.5 Voorbeelden

In deze paragraaf demonstreer ik mijn uitbreiding door een aantal voorbeelden te geven. Deze voorbeelden zijn puur ter illustratie waarbij mijn toegevoegde relaties worden toegepast.

In de voorbeelden gebruik ik een zelfontworpen klassendiagram welke een verband heeft met het ea-model. Dit klassendiagram kan enige inconsistentie bevatten en dient puur ter illustratie om te laten zien hoe de toegevoegde relaties de mogelijkheid bieden, om een klassendiagram te koppelen.

8.5.1 Voorbeeld 1



Figuur 67: Voorbeeld 1 uitbreiding ARMOR

8.5.1.1 Beschrijving

In het bovenstaande voorbeeld heb ik de volgende relatie toegepast:

- Class (UML) – Requirement (ARMOR)

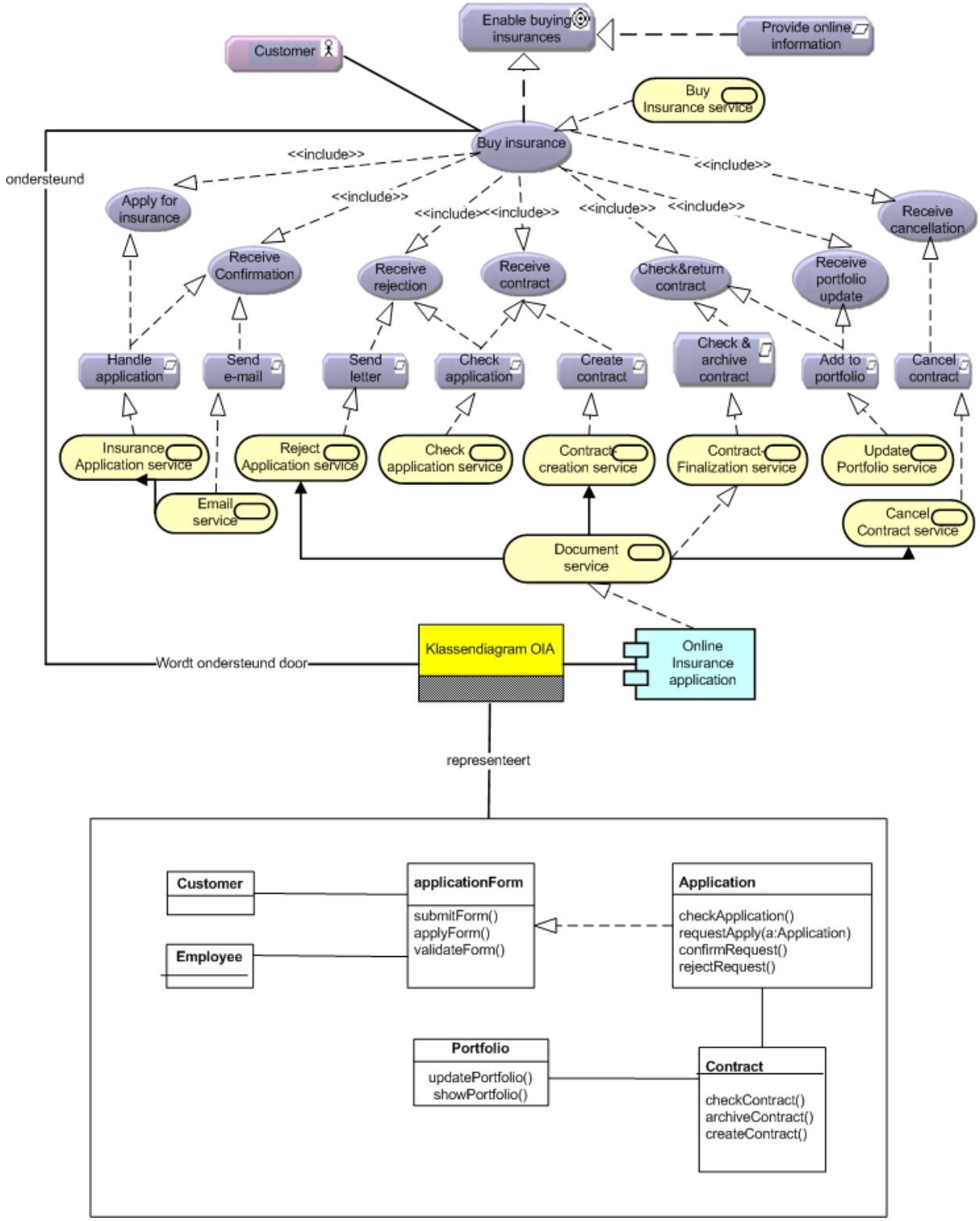
In de bovenstaande afbeelding is te zien dat de functionele systeemrequirements zijn gekoppeld met verantwoordelijke klassen. De operaties van de klassen realiseren de systeemrequirements. Ter verduidelijking geef ik onderstaande overzicht:

Requirement	Verantwoordelijke operaties	Klasse
Handle application	ApplyForm(), validateForm()	applicationForm
Send E-mail	submitForm()	applicationForm
Send letter		Application
Check application	checkApplication(), rejectRequest(), confirmRequest()	Application
Create contract	createContract	Contract
Check & archive contract	checkContract(), archiveContract()	Contract
Cancel contract	cancelContract	Contract
Add to portfolio	updatePortfolio()	Portfolio

Tabel 9: Voorbeeld 1 relaties

Op deze manier kan men snel inzicht krijgen in welke klassen en bijbehorende operaties verantwoordelijk zijn voor welke functionele requirements.

8.5.2 Voorbeeld 2



Figuur 68: Voorbeeld 2 uitbreiding ARMOR

8.5.2.1 Beschrijving

In het bovenstaande voorbeeld heb ik de volgende relaties toegepast:

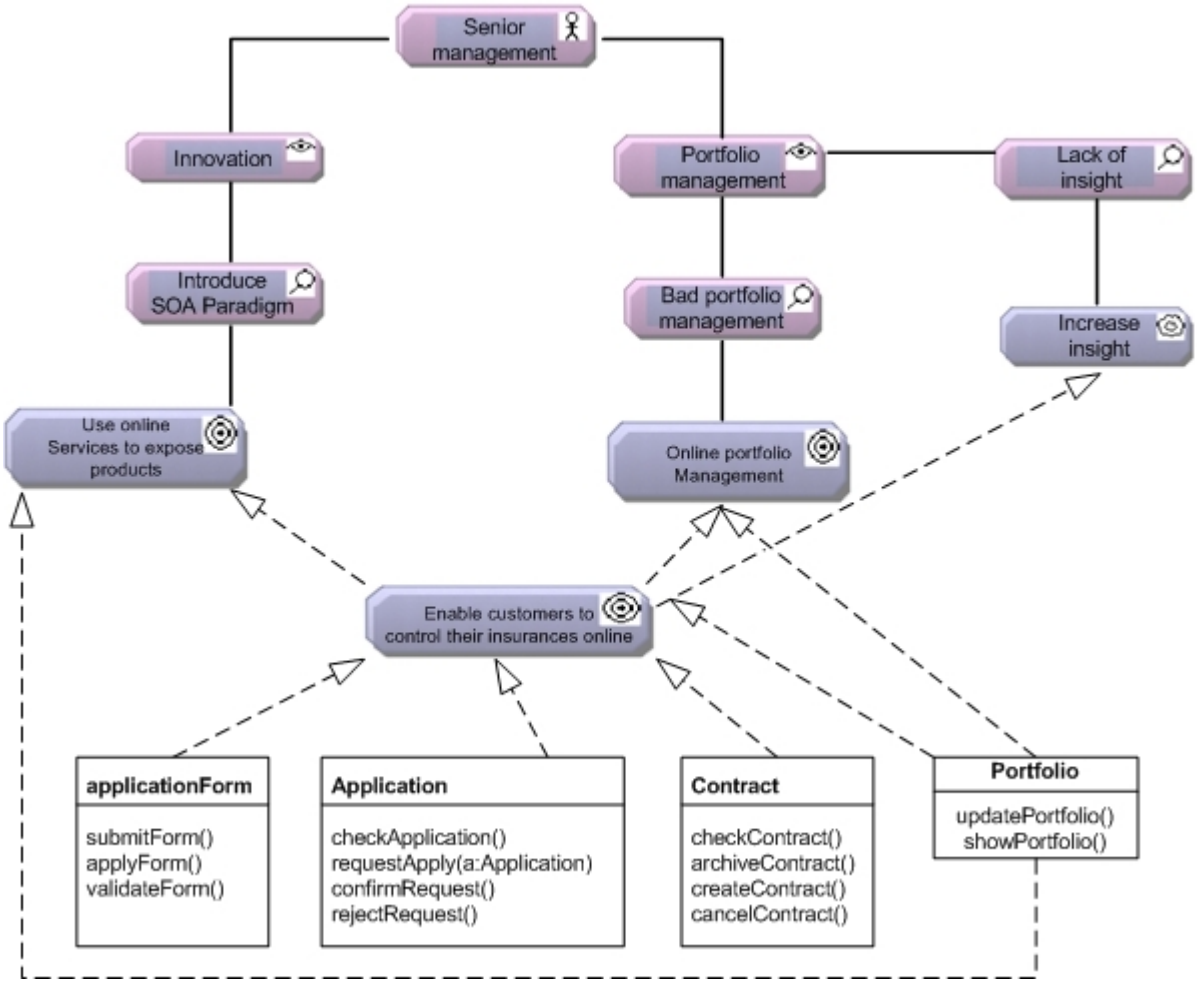
1. ClassDiagram (ARMOR) – AssociationRelation (ArchiMate)
2. AssociationRelation (ArchiMate) – StructureElement (ArchiMate)
3. ClassDiagram (ARMOR) – Class (UML)
4. ClassDiagram (ARMOR) – UseCase (ARMOR)

De eerste twee relaties worden toegepast bij de koppeling tussen het klassendiagram (Klassendiagram OIA) en het applicatiecomponent van ArchiMate (Online Insurance Application). Aangezien ik de syntax van het oorspronkelijke ARMOR meta-model heb gehandhaafd is het noodzakelijk om twee relaties te realiseren waarbij er een een koppeling wordt gemaakt met het AssocationRelation concept en het StructureElement concept. Echter zijn deze twee relaties niet zichtbaar bij het toepassen van deze relatie.

De derde relatie is de koppeling tussen het klassendiagram concept (Klassendiagram OIA) en het daadwerkelijke model van het klassendiagram. Op deze manier wordt het klassendiagram gerepresenteerd.

De vierde relatie realiseert de koppeling tussen de use-case (Buy Insurances) en het klassendiagram (Klassendiagram OIA).

8.5.3 Voorbeeld 3



Figuur 69: Voorbeeld 3 uitbreiding ARMOR

Beschrijving voorbeeld 3

In het bovenstaande voorbeeld heb ik de volgende relatie toegepast:

- Class (UML) – Goal (ARMOR)

In het model zijn drie (harde) doelen te zien, namelijk "Use online services to expose products", "Enable customers to control their insurances online" en "Online portfolio management". De klassen die verantwoordelijk zijn voor het realiseren van deze doelen, zijn met elkaar gekoppeld. Het is belangrijk om te vermelden dat het in de praktijk onmogelijk is om direct het bovenstaande model te modelleren. Het is immers noodzakelijk om doelen te operationaliseren zodat de functionele requirements bekend zijn. Deze relatie kan in het eindstadium gebruikt worden om in kaart te brengen welke klassen welke doelen realiseren.

9 Conclusie

De onderzoeksvraag van deze Master Thesis luidt:

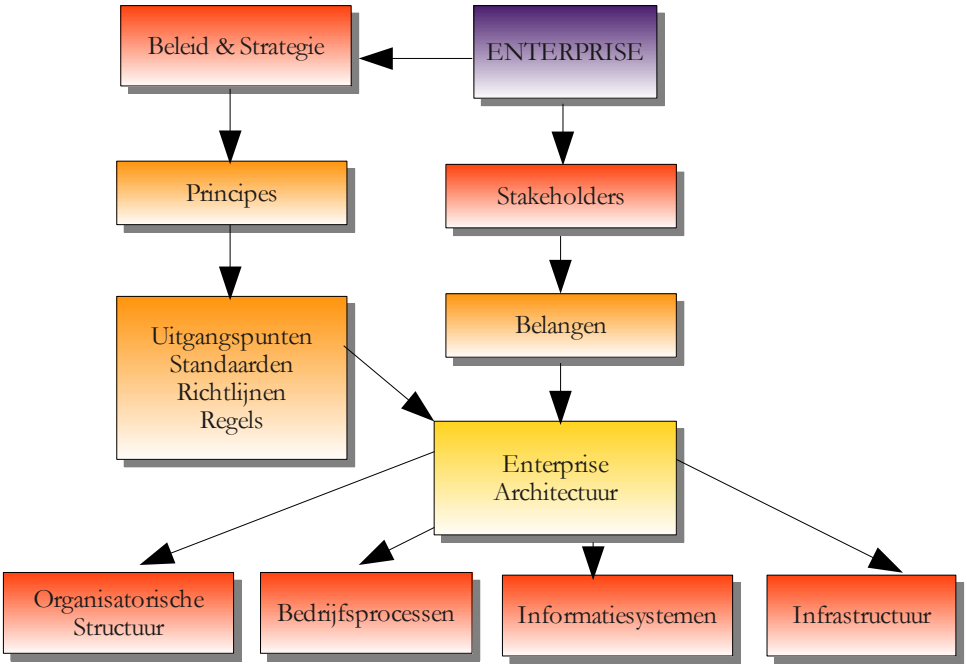
- Op welke manier is het mogelijk om de compliance tussen een enterprise-architectuur model en de bijbehorende requirements te toetsen?

Om deze onderzoeksvraag te kunnen beantwoorden heb ik onderzoek gedaan naar twee verschillende kennisgebieden, namelijk enterprise-architectuur en requirements management. De onderzoeksvraag is opgedeeld in een aantal deelvragen die ik in dit hoofdstuk telkens benoem voordat deze beantwoord worden.

Deelvraag 1: Hoe is enterprise-architectuur gepositioneerd binnen een organisatie?

Deelvraag 2: Op welke manier wordt een enterprise-architectuur beïnvloed en wordt deze ontworpen?

Enterprise-architectuur speelt zich op een hoog niveau van een enterprise af, waarbij principes en belangen van stakeholders leidend zijn voor het ontwerp van de architectuur. De principes komen voort uit het beleid en de strategie. Deze principes worden verbijzonderd naar uitgangspunten, regels, standaarden en richtlijnen. Vervolgens worden deze principes en belangen van stakeholders gebruikt voor het ontwerp en realisatie van de organisatorische structuur, bedrijfsprocessen, informatiesystemen en infrastructuur van een enterprise. Dit wordt samengevat en geïllustreerd in het onderstaande model:



Figuur 70: Invloed op en van enterprise-architectuur

Deelvraag 3: Hoe wordt er gecommuniceerd op enterprise-architectuur niveau?

Een ArchiMate-model geeft inzicht in de architectuur, in de samenhang van de verschillende lagen en de relaties tussen artefacten van deze lagen. Het framework bestaat uit 31 concepten en 12 relaties, een breed scala aan artefacten die gebruikt kunnen worden om een architectuur te visualiseren. Dat ArchiMate een goede tool is voor het modelleren van een enterprise-architectuur wordt onder andere bevestigd door het feit dat het mogelijk is om vanuit verschillende gezichtspunten te modelleren (viewpoints van stakeholders) en door het realiseren van samenhang en integratie van de verschillende lagen.

Deelvraag 4: Op welke manier is het mogelijk om requirements te herleiden?

Requirements management bestaat uit requirements engineering en requirements traceability. Requirements engineering bestaat uit zeven verschillende fases, waarbij de fases specification en validation betrekking hebben op mijn onderzoek. Ik onderscheid twee type requirements, de requirements die gebaseerd zijn op de principes en de belangen van stakeholders: **architectuurrequirements** en de requirements die opgesteld worden op basis van het architectuurontwerp voor het ontwerp van een systeem: **systeemrequirements**. Architectuur beïnvloed de functionaliteit van systemen, maar verouderde (legacy) systemen zouden ook de architectuur kunnen beïnvloeden. Op dit moment concludeer ik het volgende:

Een **requirementstaal** voor **enterprise-architectuur** moet de mogelijkheid hebben om requirements op architectuurniveau te kunnen herleiden naar requirements van systeemontwerpen. Hierdoor kan men zien wat voor een invloed een principe / strategisch doel heeft op de functionaliteit van een systeem en kan men zien wat voor een invloed een systeembeperking heeft op het ontwerp van een architectuur.

Op basis van mijn onderzoeksresultaten uit hoofdstuk 4 heb ik het requirements traceability model van Finkelstein aangepast en opgedeeld in twee verschillende processen.

Traceability-proces 1

Pre RS-traceability: het herleiden van de requirements vanuit of naar de uitgangspunten, standaarden, richtlijnen en regels die gebaseerd zijn op de principes van een enterprise.

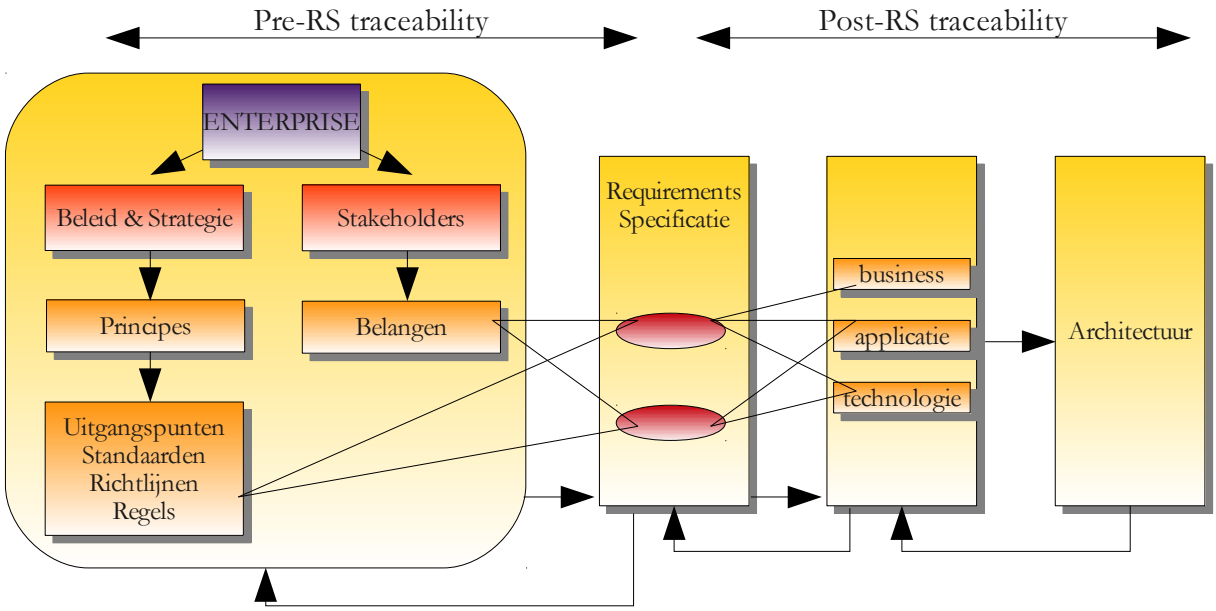
Post RS-traceability: het herleiden van de requirements vanuit of naar het ontwerp van een domein, deelarchitectuur of het complete coherente architectuurontwerp.

Dit proces maakt het mogelijk om de compliance tussen de bron en de architectuur te toetsen waarbij

men inzicht krijgt in de volgende punten:

- Welke (architectuur)requirements zijn er ontstaan op basis van de principes of belangen en stroken deze met elkaar?
- Op basis van welke (architectuur)requirements is de architectuur ontworpen en stroken deze met elkaar?
- Hoe hebben de principes en belangen het ontwerp van de architectuur bepaald?

Op deze manier kan men het architectuurontwerp onderbouwen en motiveren.



Figuur 71: Requirements Traceability 1 [FINK94] aangepast

Traceability-proces 2

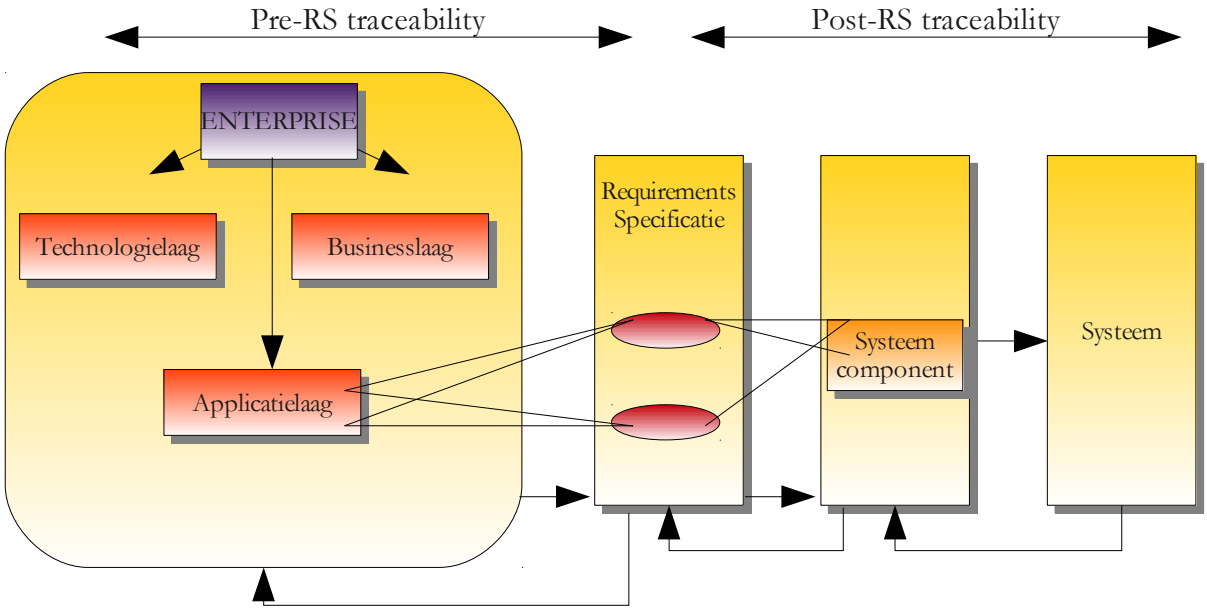
Pre RS-traceability: het herleiden van de systeemrequirements vanuit of naar de bron (architectuurontwerp).

Post RS-traceability: het herleiden van de systeemrequirements vanuit of naar het ontwerp van een applicatie component of systeemontwerp.

- Welke systeemrequirements zijn er ontstaan op basis van het architectuurontwerp en stroken deze met elkaar?
- Op basis van welke systeemrequirements is het systeem ontworpen en stroken deze met elkaar?

- Hoe heeft het architectuurontwerp het ontwerp van het systeem bepaald?

Op deze manier kan men het systeemontwerp onderbouwen en motiveren.



Figuur 72: Requirements Traceability 2 [FINK94] aangepast

Deelvraag 5: Aan welke eisen moet een requirementstaal voor enterprise-architectuur voldoen?

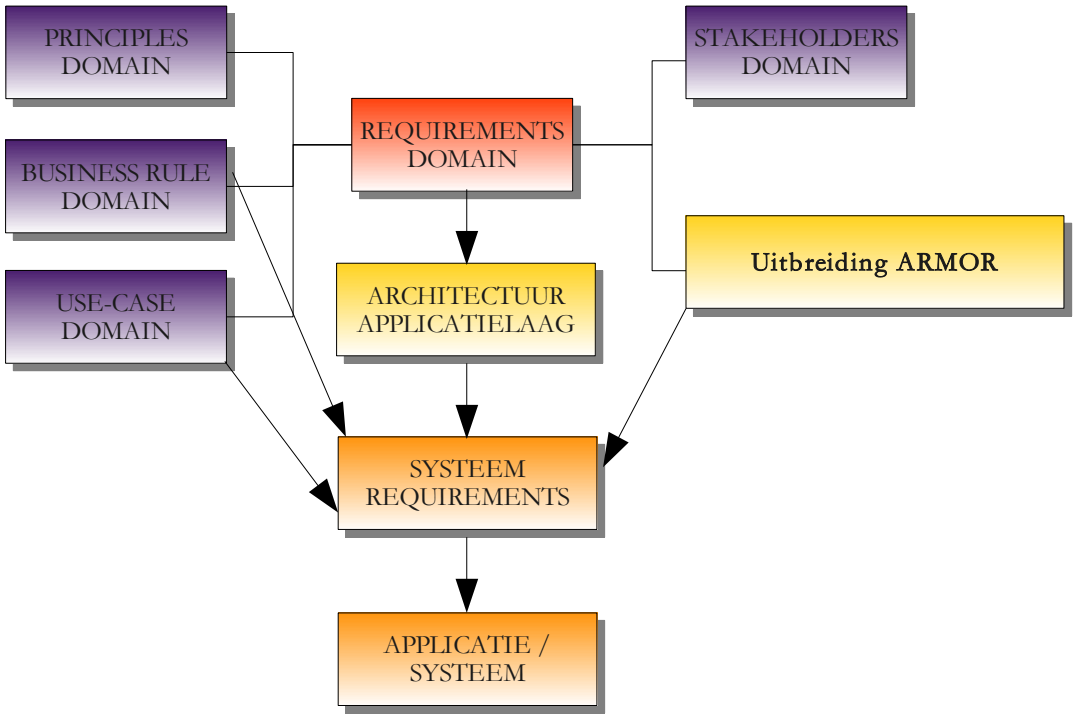
Er zal een proces plaats moeten vinden welke een abstract doel (principe, doelstelling, belang) operationaliseert naar een concrete requirement. Dit proces kan worden uitgevoerd door middel van goal-oriented requirements engineering. Goal-oriented requirements engineering wil zeggen dat men requirements management uitvoert door de doelen die op het hoogste niveau van de enterprise zijn geformuleerd, als leidraad te gebruiken voor het uitvoeren van requirements management. Twee belangrijke concepten hiervan zijn:

- doelverfijning: doelverfijning zorgt ervoor dat requirements gestructureerd uitgewerkt kunnen worden. Op deze manier is het mogelijk om requirements te herleiden en te verklaren aan stakeholders en kan men controleren of er tussen beiden sprake is van volledige compliance.
- alternatieve doelverfijning en conflictrelaties: Door alternatieven te modelleren kan men verklaren waarom er bepaalde keuzes en beslissingen zijn gemaakt. Wanneer twee doelen met elkaar conflicteren kan er gekozen worden voor een bepaald alternatief. Een dergelijk conflict kan met een conflictrelatie gemodelleerd worden.

Deelvraag 6: Voldoet ARMOR, de requirementstaal voor enterprise-architectuur aan de gestelde eisen?

ARMOR is een goal-oriented requirementstaal voor entperise-architectuur. Deze taal hanteert de filosofie dat alle architectuurrequirements voortkomen uit principes, strategische doelstellingen en belangen van stakeholders. Tevens bezit deze taal de twee bovenstaande concepten. Echter wijst mijn literatuurstudie uit dat ARMOR een zwak punt bezit, namelijk de koppeling met het operationele niveau van een enterprise.

Het modelleren binnen ARMOR beperkt zich voornamelijk in de hoge laag van de enterprise waarbij het requirements-traceability-proces 1 (zie blz. 34) uitgevoerd kan worden. ARMOR heeft de mogelijkheid om use-cases en bedrijfsregels te modelleren, wat een klein deel van het tweede requirements-traceability-proces 2 (zie blz. 35) afvangt, maar mist de koppeling met het operationele niveau. Dit wordt in de onderstaande afbeelding geïllustreerd:

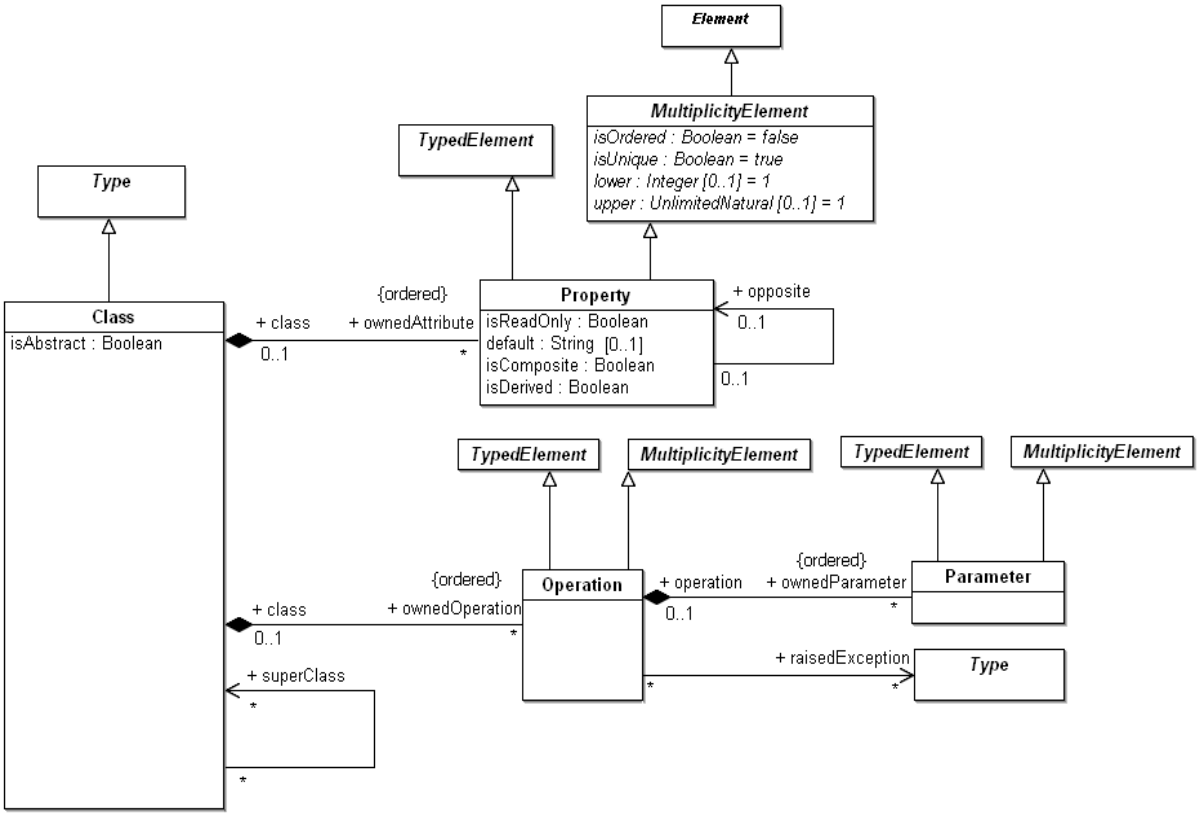


Figuur 73: Uitbreiding ARMOR

De paarse vakken zijn van ARMOR, "architectuur applicatielaag" is onderdeel van ArchiMate, de oranje vakken zijn ter verduidelijking.

De uitbreiding voor ARMOR heb ik gerealiseerd door te kiezen voor het UML klassendiagram.

Gedetailleerde beschrijvingen van artefacten uit een applicatielaag ea-model, worden doorgaans gemodelleerd in UML (Unified Modelling Language). UML is een standaard voor het modelleren en ontwerpen van systemen en applicaties. Klassendiagrammen zijn de meest voorkomende diagrammen binnen objectgeoriënteerd modelleren. Een klassendiagram geeft een verzameling klassen, interfaces en samenwerkingen (collaborations) met hun relaties weer. Een klassendiagram wordt gebruikt om het statische ontwerp van een systeem te visualiseren. Meestal wordt een dergelijk diagram gebruikt om de vocabulaire van een systeem, de samenwerkingen en (database) schema's te modelleren. Samengevat bevat een klassendiagram uit de volgende onderdelen: klassen, interfaces, samenwerkingen en relaties. UML is gebaseerd op een architectuur van vier meta-modellering niveaus. Elk niveau is gelabeld van M³ tot M° en worden meestal verwoord als het meta-metamodel, metamodel, klassendiagram en objectdiagram. Hieronder volgt het meta-model van het klassendiagram zoals deze is gedefinieerd in [OMG09, p.95].



Figuur 74: Meta-model van het klassendiagram [OMG09, p. 95]

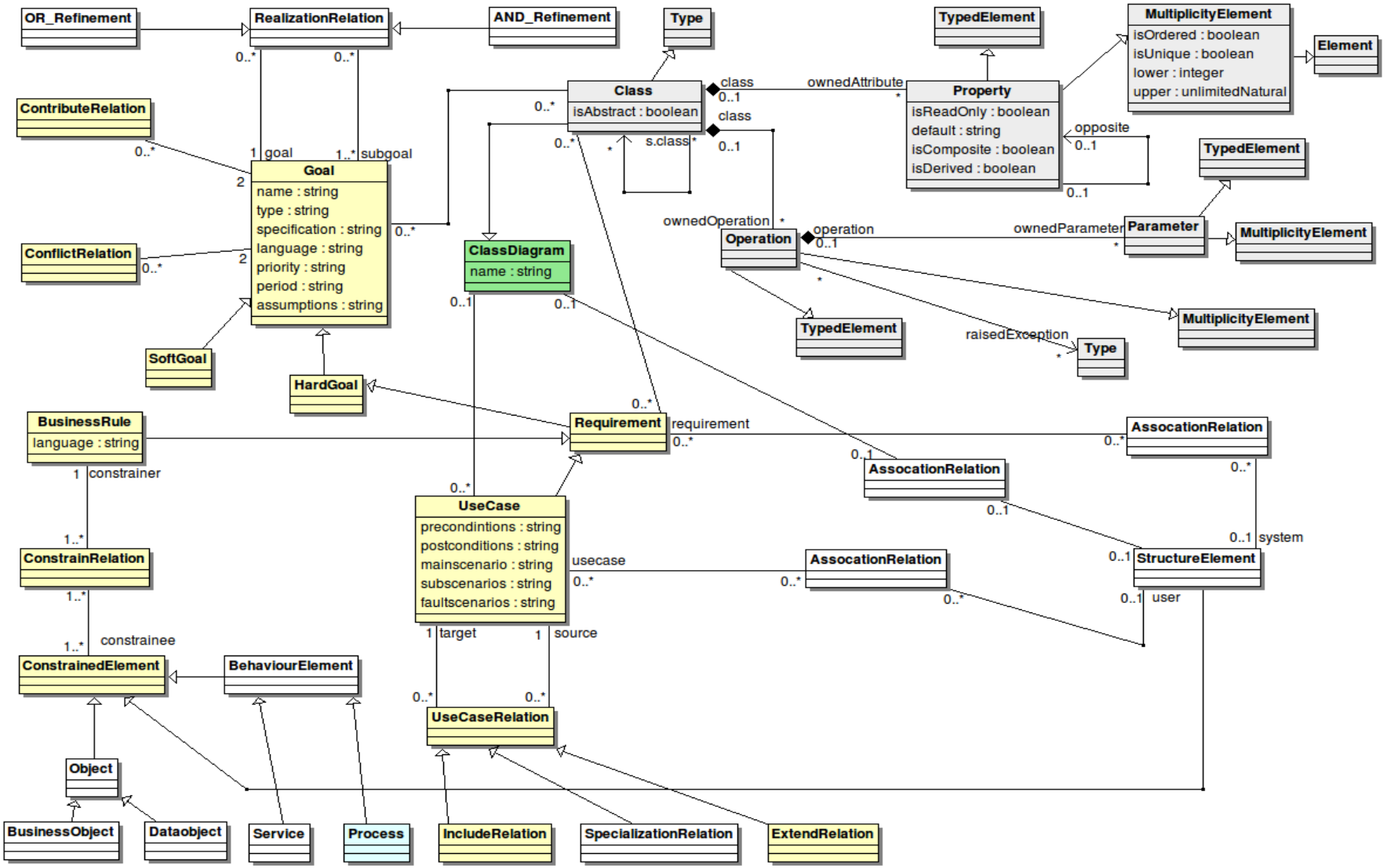
Het bovenstaande meta-model van het klassendiagram heb ik aan het ARMOR meta-model toegevoegd. Dit heb ik gedaan door relaties te realiseren tussen concepten van het UML meta-model, ARMOR meta-model en het ArchiMate meta-model. Deze relaties worden in de onderstaande tabel

weergegeven:

Concept	Concept	Relatie	Natuurlijke taal
ClassDiagram (ARMOR)	Class (UML)	Generalisatie	De klasse ClassDiagram overerft de eigenschappen van het UML meta-model.
ClassDiagram (ARMOR)	UseCase (ARMOR)	Associatie	Een use-case kan maximaal één klassendiagram ondersteunen, maar een klassendiagram kan ondersteund worden door meerdere use-cases.
ClassDiagram (ARMOR)	AssociationRelation (ArchiMate)	Associatie	Een klassendiagram kan aan nul of maximaal één AssociationRelation gekoppeld zijn. Een AssociationRelation kan aan nul of maximaal één klassendiagram gekoppeld zijn.
AssociationRelation (ArchiMate)	StructureElement (ArchiMate)	Associatie	Een AssociationRelation kan aan nul of maximaal één StructureElement gekoppeld zijn. Een StructureElement kan aan maximaal één AssociationRelation gekoppeld zijn.
Class (UML)	Requirement (ARMOR)	Associatie	Een klasse is verantwoordelijk voor nul of meerdere requirements en een requirement kan worden behaald door nul of meerdere klassen.
Class (UML)	Goal (ARMOR)	Associatie	Een goal kan door nul of meerdere klassen ondersteund worden. Een klasse kan nul of meerdere goals ondersteunen.

Tabel 10: Nieuwe relaties van de ARMOR uitbreiding

Dit resulteert een nieuw meta-model voor ARMOR:



Figuur 75: Uitgebreid meta-model ARMOR

9.1 Vervolgstudie

Het feit dat ik deze Master Thesis heb afgerond, heeft mij inzicht gegeven in interessante mogelijkheden om dit onderzoek te vervolgen.

De scope van dit onderzoek was voornamelijk gericht op de applicatielaag van ArchiMate. Koppeling met het operationele niveau van een enterprise is ook mogelijk door in te zoomen op bijvoorbeeld de bedrijfslaag. In dit veld speelt BPMN (Business Process Modeling Notation) een belangrijke rol. Figuur 16 (zie blz. 32) wijst uit dat deze modellen reeds worden gekoppeld maar dat hier tevens nog geen formele uitbreiding beschikbaar voor is. Hetzelfde geldt voor de technologielaag. SysML is een UML variant waarmee technische specificaties van hardware in kaart kunnen worden gebracht. Het koppelen van deze modellen binnen ARMOR past binnen de mogelijkheden voor een vervolgstudie.

Ik heb bewust voor het UML klassendiagram gekozen omdat dit type model inzicht geeft in de algemene functionaliteit van een systeem. Men zou ook de keuze kunnen maken om ARMOR compatibel te maken met overige UML modellen die betrekking hebben op de applicatielaag, waarbij er meer informatie kan worden gegeven over specifieke processen en onderdelen van applicaties. Denk bijvoorbeeld aan het koppelen van activiteitendiagrammen, sequencediagrammen of deploymentdiagrammen.

Er is op dit vlak nog genoeg werk te verrichten, dat wijst ook het ARMOR onderzoek uit. De aspecten “betekenis” en “waarde” uit het ARMOR-framework worden niet in het gepubliceerde rapport toegelicht en er is reeds nog geen nieuw artikel hierover gepubliceerd. Dit geldt ook voor het principles domein welke nog in de kinderschoenen staat en nog niet optimaal is geïntegreerd.

10 Bibliografie

[AURU05] Aurum en Wohlin, *Engineering And Managing Software Requirements*, Springer, 2005.

[BASS03] Bass, Clements en Kazman, Software Architecture in Practice, Second Edition, Addison Wesley, 2003.

[BERG04] Berg en Steenbergen, *Stap voor Stap naar Professionele Enterprise-Architectuur*, Ten Hagen & Stam.

[BERG07] Berg, Bosma, Dijk, Drunen, Gijsen, Langeveld, Luijpers, Nguyen, Oosting, Slagter en Willemsz, ArchiMate in de Praktijk, versie 2.0 7-10-2007.

[CHEN07] Cheng, Betty en Atlee, Research directions in requirements engineering, *In FOSE '07: 2007 Future of Software Engineering*, Washington, DC, USA, IEEE Computer Society.

[BLAA06] Blaauboer, *Requirements in Functional IT Management.* Technical Report TR-CTIT-06-07, Centre for Telematics and Information Technology, University of Twente, Enschede. ISSN 1381-3625, 2006.

[BRAY02] Bray, *An Introduction to Requirements Engineering*. Addison-Wesley, 2002.

[BOOC99] Booch, Rumbaugh, en Jacobson: The Unified Modeling Language User Guide, Addison-Wesley, 1999.

[BUIT07] Buitenhuis, *Fundamenten van het principe*, Master Thesis Radboud Universiteit, 2007

[DAFT01] Daft, *Organization Theory and Design*, Mason, South-Western Publishing, 2001.

[DARD93] Dardenne, Lamsweerde en Fickas, Goal-Directed Requirements Acquisition, *Science of Computer Programming*, Vol.20, 1993.

[DAVI90] Davis en Alan, Software Requirements Analysis and Specification, Prentice-Hall, New Jersey, 1990.

[DIET98] Dietz, De Informatie-architect op zijn plaats gezet, *Informatie, maandblad voor de informatievoorziening*, 1998.

[ENCY09] Online Encyclopedie, website, 2009, http://www.encyclo.nl.

[FINK94] Gotel en Finkelstein, A*n analysis of the requirements traceability problem*, in Proceedings of ICRE94, 1st International Conference on Requirements Engineering,

Colorado Springs, Co, IEEE CS Press, 1994.

[FINK00] Finkelstein en Emmerich. *The future of requirements management tools*, Information Systems in Public Administration and Law, 2000.

[BOSM05] Bosma, Jonkers en Lankhorst, Inleiding in de ArchiMate-taal, versie 1.1 24-10-2005.

[HOOG04] Hoogervorst, Enterprise Architecture: Enabling Integration, Agility and Change, Published in: International Journal of Cooperative Information Systems, Vol. 13, No. 3, 2004.

[HOOG07] Hoogevorst, *Enterprise governance & architectuur*, Den Haag, ICT-Bibliotheek, 2007.

[IACO04] Iacob, Jonkers en Wiering, *Towards a UML profile for the ArchiMate language*, Telematica Instituut, 17 December 2004.

[LAMS01] Lamsweerde, *Goal-Oriented Requirements Engineering: A Guided Tour*, Proceedings of the Fifth International Symposium on Requirements Engineering 2001 IEEE.

[LANK05] Lankhorst, Enterprise Architecture At Work, Modelling, Communication and Analysis, Springer-Verlag Berlin Heidelberg, 2005.

[LANK04] Lankhorst, van der Torre, ter Doest en Stam, *ArchiMate verbindt architectuurdomeinen*, Informatie, Themanummer alignment, april 2004.

[LOUC95] Loucopoulos en Kavakli, *Enterprise Modelling and the Teleological Approach to Requirements Engineering*. International Journal of Intelligent and Cooperative Information Systems, 1995.

[LUIJ07] Luijpers, Whitepaper Project Start Architectuur (PSA), Sogeti, November 2007.

[MAIE01] Maier, Emery, en Hilliard, Software Architecture: Introducing IEEE Standard 1471, IEEE Computer, April 2001, Vol. 34-4.

[MELA05] Melaard, Ondernemingstypering uit architectuurcontext, Master Thesis Digitale Architectuur, Radboud Universiteit Nijmegen, 24-08-2005.

[OMG08] Object Management Group, *Semantics of Business Vocabulary and Business Rules (SBVR)*, v1.0 , 2008-01-02.

[OMG09] Object Management Group, OMG Unified Modeling Language (OMG UML), Infrastructure, v2.2, 2009-02-04.

[PRESS05] Pressman, Software Engineering, A Practitioner's Approach, Sixth Edition, McGraw-Hill, 2005.

[QUAR091] Quartel, Engelsman, Jonkers en van Sinderen, *A goal-oriented requirements modelling language for enterprise architecture*, 2009, IEEE.

[QUAR092] Quartel, Engelsman, Jonkers en van Sinderen, *A goal-oriented requirements modelling language for enterprise architecture*, ARMOR for requirements modelling, 2009, Novay BizzDesign.

[RIJS04] Rijsenbrij, Architectuur in de Digitale Wereld (Versie Nulpuntdrie), *Inaugurale Rede*, Radboud Universiteit Nijmegen, 2004.

[RIJS05] Rijsenbrij, Kanttekeningen bij de 'Architectuur in de Digitale Wereld' (Versie Nulpuntzes), Radboud Universiteit Nijmegen, 2005.

[SOET04] Soetekouw, Corporate Architecture, *PrimaVera Working Paper*, Universiteit van Amsterdam, 2004.

[THEO091] The Open Group, What is Enterprise Architecture, 28-09-2009, http://opengroup.co.za/enterprise-architecture.

[THEO092] The Open Group, ArchiMate 1.0 Specification, 2009.

[TRUI05] Truijens, Architectuur: Werkterrein van Ingenieurs of Speelveld van Ontwerpers?, *PrimaVera working paper*, Universiteit van Amsterdam, 2005.

[WAGT01] Wagter, van den Berg, Luijpers en Steenbergen, *DYA: Snelheid en Samenhang in Business-en ICT-architectuur*, Tutein Nolthenius, 2001.

[VAND09] Van Dale Woordenboek, website, 2009, http://pro.vandale.nl

[ZAVE96] Zave, Classification of Research Efforts in Requirements Engineering, *ACM Computing Surveys,* 29(4), 1997.

[ZACH96] Zachman, Enterprise Architecture: the Issue of the Century, Zachman International, 1996.

[ZAV97] Zave en Jackson, "Four Dark Corners of Requirements Engineering", *ACM Transactions on Software Engineering and Methodology*, 1997.