Model Driven Architecture

Specificeren van software functionaliteit

Afstudeerscriptie Master Software Engineering, Faculteit der Natuurwetenschappen, Wiskunde en Informatica, Universiteit van Amsterdam

Auteur: **Wilfred Belo**

Begeleider: **J.F.J. Branderhorst (Ordina)**

Afstudeerdocent: **Alban Ponse**

Opdrachtperiode: 29 maart t/m 25 juni 2004

Voorwoord

Voor u ligt de scriptie van het afstudeerproject van Wilfred Belo in opdracht van de Universiteit van Amsterdam, uitgevoerd bij Ordina. Ter afsluiting van de Master Software Engineering aan de Universiteit van Amsterdam is een afstudeerperiode van 3 maanden in het onderwijsprogramma opgenomen. Het doel van deze afstudeerperiode is dat de student aantoont dat hij voldoet aan de eindtermen van de Master Software Engineering.

Deze afstudeerperiode heb ik kunnen volbrengen bij Ordina, één van de grotere ICT-dienstverleners in Nederland.

Mijn dank gaat uit naar mijn bedrijfsbegeleider de heer J.F.J. Branderhorst voor zijn begeleiding en belangstelling. Tot slot wil ik de mensen bedankt die mij op wat voor wijze dan ook hebben gesteund in de totstandkoming van deze scriptie. Hieronder behoren ook de docenten aan de Universiteit van Amsterdam.

Wilfred Belo
Juni 2004

Samenvatting

Het onderzoek richt op de vraag in hoeverre J2EE software functionaliteit gespecificeerd kan worden op het PIM-abstractieniveau met behulp van Model Driven Architecture (MDA). Dit is een methodiek voor het ontwikkelen van software, waarbij het gebruik van formele modellen tijdens het software ontwikkelproces centraal staat.

Er zijn drie hoofdtypen van J2EE software functionaliteit onderscheiden, namelijk business logic / rules, gegevensopslag / -verwerking en communicatie / gegevensuitwisseling. Vanuit de onderscheiden typen software functionaliteit zijn hypothesen opgesteld over de specificeerbaarheid ervan op het PIM-abstractieniveau. Er is getracht deze hypothesen te toetsen met gebruik van OptimalJ.

In de praktijk is gebleken dat J2EE software functionaliteit (nog) niet in zijn totaliteit gedefinieerd kan worden op het PIM-abstractieniveau. Meer dan de helft van de hypothesen kunnen momenteel niet worden getoetst met behulp van OptimalJ, aangezien deze MDA ontwikkelomgeving een aantal technieken niet ondersteunt. Voor het specificeren van het merendeel van de onderscheiden typen software functionaliteit biedt OptimalJ specifieke voorzieningen, die geen standaard binnen het MDA-raamwerk zijn.

Wel kan OptimalJ buiten de statische structuur van een bedrijfsaspect (onderdeel van het type software functionaliteit business logic) business rules vastleggen door middel van de *business expression library*, wat een OptimalJ-specifieke voorziening is voor het beheren van business rules. Ook kan er gebruik worden gemaakt van het *integration model*, waarmee interfaces worden geleverd naar externe of bestaande systemen aan de hand van interface definities.

De *business expression library* bestaat naast de abstractieniveaus PIM en PSM. Het gebruik van de business rules uit deze *business expression library* moet handmatig geprogrammeerd worden.
OptimalJ biedt geen voorziening waarbij de dynamische aspecten van een software systeem gespecificeerd kunnen worden anders dan door het schrijven van code. Condities en integriteits constraints worden op het PSM abstractieniveau vastgelegd, in tegenstelling tot het PIM-abstractieniveau.

Inhoudsopgave

1. Aanleiding

1.1. Inleiding

Dit onderzoek is uitgevoerd ter afsluiting van de Master Software Engineering aan de Universiteit van Amsterdam.

De probleemstelling van het onderzoek en de structuur van deze scriptie komen respectievelijk aan de orde in paragraaf 1.2 en 1.3.

1.2. Probleemstelling

Het onderzoek betreft de mogelijkheid tot het specificeren van functionaliteit van een J2EE [C15] systeem m.b.v. Model Driven Architecture (MDA). Model Driven Architecture (MDA) is een methodiek voor het ontwikkelen van software, waarbij het gebruik van formele modellen tijdens het software ontwikkelproces centraal staat. Voor een verdere toelichting van het MDA-raamwerk zie hoofdstuk 2. De probleemstelling is tot stand gekomen door interesse vanuit Ordina in het toepassen van MDA in het J2EE ontwikkeltraject.

Voor het daadwerkelijk toepassen van MDA in het J2EE ontwikkeltraject van Ordina, zal er onderzocht moeten worden of MDA in de praktijk bruikbaar is. Er is relatief veel bekend over het realiseren van de architectuur en onderliggende techniek van een J2EE systeem met behulp van MDA, maar over het specificeren van software functionaliteit is meer onduidelijkheid.

Het aspect van het specificeren van software functionaliteit met behulp van MDA is het onderwerp van dit afstudeeronderzoek.

Bovenstaande probleemstelling heeft geleid tot de volgende centrale vraagstelling:

In hoeverre is het mogelijk om software functionaliteit op het hoogste abstractieniveau van MDA (het Platform Independent Model) te specificeren, zodanig dat het mogelijk is deze functionaliteit te transformeren naar een werkend J2EE software systeem?

Het onderzoek is in de volgende delen onderverdeeld:

- Classificeren van software functionaliteit, gericht op J2EE systemen
- Hypothesen m.b.t. de specificeerbaarheid per type functionaliteit op PIM-abstractieniveau
- Mogelijkheden verkennen van het uitwerken van een case die typen functionaliteit afdekt
- Toetsingsmogelijkheden en toetsing van de opgestelde hypothesen aan de hand van mogelijkheden uit praktijk

Zie bijlage A voor het plan van aanpak.

1.3. Leeswijzer

De opbouw van de scriptie is als volgt. Het Model Driven Architecture raamwerk wordt in hoofdstuk 2 besproken. In hoofdstuk 3 wordt ingegaan op software functionaliteit en de onderverdeling hiervan. Tevens worden aan de hand van de onderscheiden typen functionaliteit hypothesen opgesteld en getoetst. Conclusies en aanbevelingen met betrekking tot de afstudeeropdracht komen in hoofdstuk 4 aan de orde. Tot slot wordt in hoofdstuk 5 een evaluatie gegeven over de afstudeerperiode.

2. Het MDA-raamwerk

2.1. Inleiding

In het vorige hoofdstuk is ingegaan op de achtergrond van de afstudeeropdracht. In dit hoofdstuk wordt ingegaan op het Model Driven Architecture raamwerk.

Model Driven Architecture (MDA) is een methodiek voor het ontwikkelen van software, gedefinieerd door de Object Management Group (OMG). MDA is nog in ontwikkeling en staat pas in de "kinderschoenen". MDA belooft een paradigmaverschuiving in het ontwikkelen van software, vergelijkbaar met de stap van assembly naar een hogere programmeertaal. De methodiek scheidt de specificatie van systeem functionaliteit van de implementatie op een specifiek platform. In het MDA-raamwerk staat het gebruik van formele modellen tijdens het software ontwikkelproces centraal. Het ontwikkelproces wordt aangestuurd door het modelleren van het software systeem.

2.2. Modellen

Een model in het MDA-raamwerk is een representatie van functies, structuur en gedrag van een systeem en is geschreven in een taal met precies gedefinieerde syntax en semantiek.

De omgeving waarin het software systeem wordt gebruikt wordt gemodelleerd in een Computation Independent Model (CIM), ook wel domein- of businessmodel genoemd. Een CIM verbergt veel of alle informatie over het gebruik van een software systeem en is onafhankelijk van hoe een systeem wordt geïmplementeerd. De requirements van een software systeem worden afgeleid uit het gedeelte van het model dat de software moet ondersteunen. Gedeelten van een CIM kunnen worden ondersteund door software systemen, maar het CIM zelf blijft software onafhankelijk.

Het MDA-raamwerk omvat drie abstractieniveaus van een software systeem, te weten een platformonafhankelijk model (Platform Independent Model, PIM), één of meer platform specifiek modellen (Platform Specific Model, PSM) en code. De drie abstractieniveaus zijn formele modellen, dus modellen die door computers begrepen kunnen worden.

Een PIM beschrijft de werking van een systeem, maar verbergt hierbij de details van de specifieke implementatie van de functionaliteit van een systeem. In het PIM wordt dat gedeelte van een systeem beschreven dat niet aan verandering onderhevig is bij een wijziging naar een specifiek platform. OMG [C1] hanteert de volgende definitie van platform:

"*A platform is a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns, which any application supported by that platform can use without concern for the details of how the functionality provided by the platform is implemented.*"

Voorbeelden van platforms zijn: middleware, besturingssystemen, .NET, XML/SOAP, CORBA, DCOM, Linux, J2EE, IBM DB2, EJB, etc. Bij platforms is er dus sprake van verschillende abstractieniveaus.

Een PIM heeft de kwaliteit dat het model onafhankelijk is van specifieke mogelijkheden van een bepaald type platform. Platformonafhankelijkheid van een model is contextafhankelijk in die mate dat een model zowel platformafhankelijk als –onafhankelijk kan zijn, afhankelijk vanuit welk oogpunt er naar wordt gekeken. Een PIM bevat een mate van platformonafhankelijkheid als het geschikt is voor verschillende platforms van hetzelfde type. Een voorbeeld is het onafhankelijk zijn van besturingssysteem, maar het afhankelijk zijn van het J2EE platform. Ook kunnen er in het PIM aannames worden gemaakt over de beschikbaarheid van middleware, maar komt er in het model niet naar voren van welk type middleware (EJB, XML/SOAP, etc) gebruik wordt gemaakt.

Wat als een PIM beschouwd kan worden hangt af van het type platform wat de MDA-gebruiker in gedachten heeft. Wat telt als een platform is relatief aan het doel van de gebruiker. Zo is voor veel MDA-gebruikers bijvoorbeeld middleware een platform, terwijl voor een middleware-ontwikkelaar het besturingssysteem het platform is. Een platformonafhankelijk model van middleware kan zeer

platformspecifiek aandoen vanuit het oogpunt van een applicatie ontwikkelaar die vanuit een hoger abstractieniveau naar het model kijkt. Voor de ene ontwikkelaar kan een PSM de PIM zijn voor een andere ontwikkelaar. De grens tussen platformafhankelijkheid en –onafhankelijkheid is hierdoor niet eenvoudig te trekken aangezien het geheel afhangt vanuit welk oogpunt ernaar wordt gekeken.

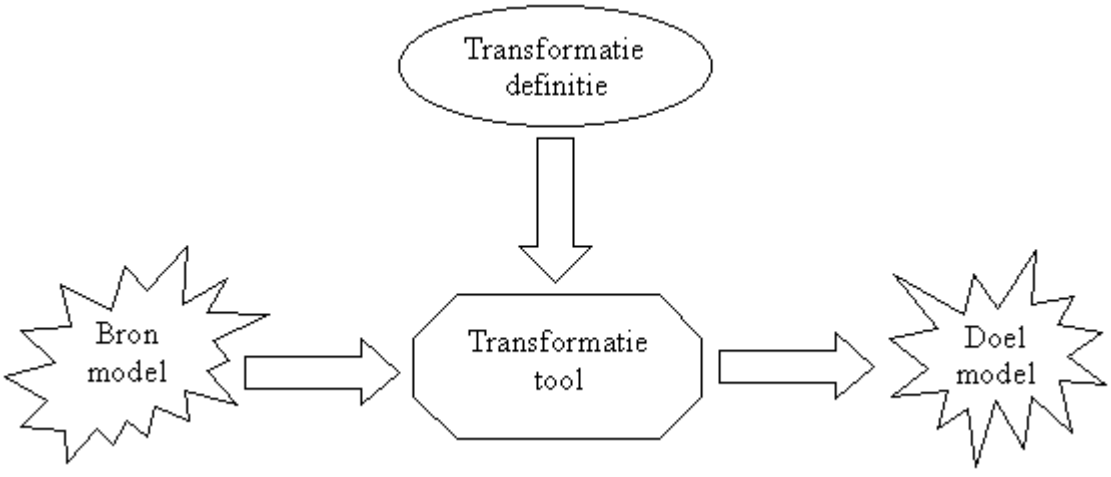
Een PIM wordt getransformeerd naar één of meer PSMs. Voor elk specifiek technologie platform wordt een aparte PSM gegenereerd. Een PSM combineert de specificaties vanuit het PIM met de details die specificeren hoe het systeem gebruik maakt van een specifiek type platform (zoals EJB, CORBA, SQL database, etc). Een PSM beschrijft hoe het systeem wordt gerealiseerd met de technische details die specifiek zijn voor een platform en is het tussenliggende abstractieniveau tussen de PIM en de broncode. De broncode is het laagste abstractieniveau van een software systeem en is zeer platform specifiek.

2.3. Transformaties

Centraal in het MDA-raamwerk staan de (automatische) transformaties tussen formele modellen, namelijk tussen PIM(s), PSM(s) en code. Een transformatie is de automatische generatie vanuit een doelmodel naar een bronmodel volgens een bepaalde transformatiedefinitie. Een transformatiedefinitie bestaat uit een verzameling transformatieregels die beschrijven hoe een model in de brontaal getransformeerd kan worden naar een model in de doeltaal. Een transformatieregel is een beschrijving van hoe bepaalde constructies uit een brontaal worden getransformeerd naar constructies in de doeltaal [A1].

De modellen die in het transformatieproces een rol spelen, het bronmodel en doelmodel, kunnen zowel in dezelfde als in een verschillende taal zijn geschreven. Een voorbeeld van het transformeren van een model naar een model geschreven in dezelfde taal is het herstructureren van een model of stuk code.
Een taal waarin transformatiedefinities worden geschreven wordt een transformatiedefinitietaal genoemd. Hiervoor is momenteel geen standaard gedefinieerd. Een transformatiedefinitietaal is een metataal (net zoals UML) en is zelf ook geschreven in een meta-metataal. Doordat alle metamodellen zijn gemaakt vanuit constructies uit de MOF, kunnen alle metamodellen generiek worden gemanipuleerd door de transformatiedefinities doordat de metamodellen universeel zijn.

Transformaties worden binnen het MDA-raamwerk door tools uitgevoerd. Aangezien het MDA-raamwerk nieuw en in ontwikkeling is, zijn er nog geen tools aanwezig die het transformatie proces volledig ondersteunen. De ontwikkelaar zal nog handmatig de getransformeerde PSM en/of code moeten uitbreiden [A1]. Figuur 2-2 geeft het transformatieproces binnen het MDA-raamwerk schematisch weer.



Figuur 2-1 Het transformatieproces binnen het MDA-raamwerk

Er kunnen 3 soorten transformaties worden onderscheiden, namelijk:

- Transformaties van en naar modellen van hetzelfde abstractieniveau
  Het herstructureren of het uitbreiden met extra informatie van een model (of code) kan worden beschreven door een transformatiedefinitie tussen modellen op hetzelfde abstractieniveau.
- Transformaties van een model op een hoger abstractieniveau naar een model op een lager abstractieniveau (forward engineering)
  Dit is de meest gebruikelijke transformatie en tijdens deze transformatie wordt een model getransformeerd naar een meer platformspecifiek model. Het transformeren van een PIM naar één of meer PSMs wordt op het moment nog niet breed ondersteund door tools. Er is meer ondersteuning van tools voor het transformeren van een PSM naar code, aangezien deze tools al op de markt waren voordat het MDA-raamwerk bekend werd.
- Transformaties van een model op een lager abstractieniveau naar een model op een hoger abstractieniveau (reverse engineering)
  Veranderingen in een model op lager niveau worden doorgevoerd naar een model op hoger niveau (zoals van code naar PSM en van PSM naar PIM). Deze vorm van transformeren zou kunnen worden toegepast bij het modelleren van legacy systemen, maar wordt in de praktijk nog nauwelijks ondersteund.

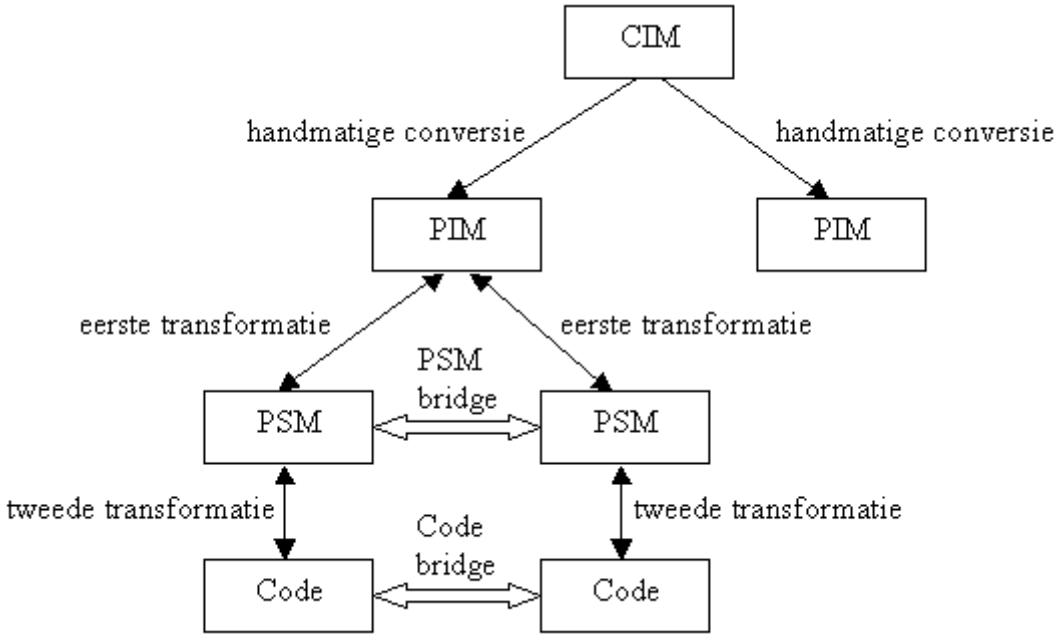
Aangezien op dit moment elke fabrikant van MDA-tools zijn eigen transformatiedefinitietaal gebruikt voor het definiëren van transformatiedefinities en deze bovendien zijn ingebouwd in de tool zelf, zijn deze niet uitwisselbaar. Dit is volledig te wijten aan het ontbreken van een standaard transformatiedefinitietaal. Vanwege de grote afhankelijkheid van MDA van transformatiedefinities en de tool-ondersteuning is er veel baat bij een standaard. Ook dit gebied is volop in ontwikkeling en één van de voorgestelde talen bij de OMG is de QVT (Query, Views and Transformations, zie bijlage B).

Flexibele tools bieden de mogelijkheid om een definitietaal te kiezen en aan de hand daarvan modellen te transformeren. Aangezien de tool-ondersteuning van het transformeren niet optimaal is kan het potentieel van MDA nog niet volledig worden benut.

Het MDA-proces lijkt erg veel op het traditioneel ontwikkelen van software. Traditioneel worden de transformaties van model naar model of van model naar code voornamelijk met de hand gedaan. Transformaties binnen het MDA-raamwerk worden altijd door tools verricht. Het transformeren van een PSM naar code is niets nieuws, maar het automatisch transformeren van PIM naar PSM(s) is dat wel.

De requirements van een systeem worden aan de hand van het Computation Independent Model afgeleid. De transformaties tussen de (formele) modellen PIM, PSM en code geschiedt automatisch. Er kunnen tevens *bridges* tussen de platformspecifieke modellen worden gegenereerd, zodat er tussen verschillende platforms communicatie mogelijk is (interoperabiliteit tussen platforms). Hierbij kan worden gedacht aan een (communicatie-) bridge tussen een relationeel database model en EJB model en tussen een web model en een EJB model. Een bridge wordt automatisch gegenereerd door een MDA tool door twee transformaties (voor het genereren van twee platforms, PSMs) te combineren tot een nieuwe transformatie.

Zo kan een transformatie van een UML klassendiagram naar een EJB specifiek model en een andere transformatie van een UML klassendiagram naar een CORBA specifiek model worden gecombineerd (en aangepast) zodat de transformatie van een PIM naar PSM een model van een bridge oplevert voor de communicatie tussen het EJB en CORBA platform. Figuur 2-2 geeft de samenhang tussen de diverse modellen weer.



Figuur 2-2 Samenhang tussen abstractieniveaus

De vertaling van het CIM naar één of meer PIMs geschiedt handmatig. Automatische afleiding van een PIM vanuit een CIM is niet geheel mogelijk, aangezien de keuzen van welke gedeelten uit een CIM moeten worden ondersteund door software, altijd door een mens gemaakt moeten worden [A1]. Het CIM verbergt de details van de structuur en werking van een systeem [C1]. Het PIM richt zich op de werking van het systeem, zodat het CIM te weinig informatie bevat voor een volledige transformatie naar een PIM. Het systeem wordt in het CIM als een black box beschouwd, in tegenstelling tot het PIM. Een automatische transformatie van het CIM naar het PIM is alleen mogelijk als het CIM wordt gemodelleerd met een formele taal, die dezelfde concepten heeft als de taal waarin het PIM (of PSM) is gemodelleerd (dat wil zeggen dat de talen bestaan uit constructies van dezelfde meta-metataal, MOF). Door het formaliseren van het CIM wordt er het risico gelopen dat het model niet goed wordt begrepen door niet-technische bedrijfsmensen. Met de proposal UML4MDA [B4, B5, C14] van de UML 2.0 Infrastructure RFP [C11] kan een nieuwe taal worden gedefinieerd gebaseerd op UML 2.0. Deze uitbreiding op UML 2.0 zou gebruikt kunnen worden als CIM taal [B4, B5]. Zodra het CIM in het MDA-proces is opgenomen, kunnen veranderingen in de PIM/PSM/code die van invloed zijn op het bedrijfsproces, worden gevalideerd door middel van het CIM.

Voor elk systeem dat een gedeelte uit een CIM ondersteund wordt een PIM ontwikkeld.

2.4. Metaniveaus

Een CIM kan geschreven worden in elke (modelleer)taal. Een PIM en PSM zijn geschreven in een formele modelleertaal, waarvoor UML op dit moment de de facto standaard is. Hoewel het gebruik van UML wordt bemoedigd door OMG is UML niet onlosmakelijk verbonden aan MDA, de eis is namelijk dat de te gebruiken talen binnen MDA geschreven zijn in Meta Object Facility (MOF). MOF is een metamodelleertaal, waarin alle andere talen worden gedefinieerd. Een metamodel definieert de abstracte syntax van een verzameling van modelleerconstructies en is de basis van een modelleertaal. Door het schrijven van modelleertalen (zoals UML) in een metataal (zoals MOF) wordt gegarandeerd dat alle talen op een standaard manier zijn opgebouwd en daardoor alom te begrijpen zijn.
OMG maakt gebruik van een vier-lagen architectuur voor zijn standaarden, die ook een rol spelen binnen het MDA-raamwerk, namelijk: M0, M1, M2 en M3. De M0 laag bestaat uit instanties van klassen (objecten), de M1 laag bestaat uit modellen van het software systeem, de M2 laag bestaat uit metamodellen en de M3 laag bestaat uit meta-metamodellen. Elk element op een lagere laag is een

instantie van een element op de hogere laag. Zo zijn alle modelleer talen (M2 laag, zoals UML, CWM, etc.) instanties van de MOF (M3 laag). Onderstaande tabel geeft de relaties weer tussen de metaniveaus. In bijlage B worden diverse technologieën besproken die relevant zijn binnen de context van MDA.

Metaniveau	Omschrijving	Elementen
M3	MOF, verzameling van constructies voor definiëren van metamodellen	MOF klasse, MOF attribuut, MOF Associatie, MOF operatie, etc.
M2	Metamodellen (UML, CWM, etc.), bestaande uit instanties van MOF constructies	UML klasse, UML associatie, UML attribuut, etc. CWM tabel, CWM kolom, etc.
M1	Modellen, bestaande uit instanties van M2 metamodel constructies	Klasse "Order", klasse "Klant", attribuut "naam" Tabel "Persoon", Tabel "Artikel", etc.
M0	Objecten en data, de instanties van M1 model constructies	Order 43123, Artikel 8RB31, etc.

Tabel 2-1 MDA metaniveaus

2.5. Commentaar

Het MDA-raamwerk is nog in ontwikkeling en is op een aantal gebieden nog niet uitgewerkt zoals het genereren van bridges, standaardiseren van transformaties, reverse engineering, het transformeren en koppelen van/uit verschillende UML modellen/diagrammen en het definiëren van de dynamische aspecten van een software systeem.

De term architectuur in de naam Model Driven Architecture is niet geheel op haar plaats. MDA is gelijk aan Model Driven Development (MDD) met het verschil dat er onderscheid wordt gemaakt tussen abstractieniveaus (PIM en PSM) en dat er wordt gebouwd op de technieken/standaarden van OMG. Bovendien is alleen UML niet geschikt om een architectuur van een systeem te ontwerpen, aangezien UML niet een taal is die door iedere stakeholder wordt begrepen.

Het principe van MDA is niet nieuw: het genereren van code aan de hand van modellen bestaat al jaren. Toch verschilt het MDA-raamwerk (in theorie) van klassieke codegeneratoren:

- De gebruiker is niet gebonden aan een specifieke vendor. Het verschil met klassieke CASE tools is dat het met MDA niet uitmaakt van welke MDA ontwikkelomgeving er gebruik wordt gemaakt. Zolang de standaards worden gehanteerd is alles (zoals modellen, transformatiedefinities) uitwisselbaar met diverse MDA ontwikkelomgevingen.
- MDA heeft een open structuur. De gebruiker heeft invloed op het proces van transformaties. De interne werking van een codegenerator is vaak een blackbox voor de gebruiker, in tegenstelling tot het gebruik van volwassen MDA ontwikkelomgevingen.
- De mogelijkheid tot roundtrip engineering, waardoor aangepaste code na generatie behouden blijft en wordt doorgevoerd naar de hogere abstractieniveaus. De code die wordt gegenereerd door klassieke codegenerators kan niet aangepast worden zonder dat de gewenste werkwijze wordt aangepast.

De MDA-tools/ontwikkelomgevingen bepalen voor een groot deel het succes van MDA. Doordat het MDA-raamwerk nieuw en volop in ontwikkeling is hebben verscheidene vendors een eigen invulling gegeven aan de wijze waarop het MDA-raamwerk is geïmplementeerd, zoals het wel/niet onderscheid maken tussen PIM/PSM en het niet kunnen beïnvloeden van transformaties. Het is ook nog de vraag in hoeverre de verscheidene MDA ontwikkelomgevingen gaan voldoen aan de technieken en standaards zoals die door OMG worden voorgeschreven.

Het MDA-raamwerk wordt beter benut zodra het CIM bij het (automatische) transformatieproces wordt betrokken. Wijzigingen in het PIM en/of PSM kunnen op deze manier worden gevalideerd door niet-technische bedrijfsmensen. Door het betrekken van het CIM bij MDA zal het op te leveren software systeem beter aansluiten bij het bedrijfsproces van de klant. Er zijn ontwikkelingen gaande op dit gebied, zoals UML4MDA [B3, C14].

Door de vele technieken die binnen het MDA-raamwerk gebruikt (kunnen) worden is het op een optimale gebruik maken van MDA complex, wat leidt tot een hoge leercurve. Het vergt een investering voordat er binnen een bedrijf optimaal van MDA gebruik gemaakt kan worden.

Op dit moment is MDA voornamelijk gericht het J2EE en .NET in mindere mate, wat vergelijkbare raamwerken zijn. Het is afwachten hoe MDA zich verder ontwikkeld op andere gebieden van softwareontwikkeling dan multitier web-based applicaties.

3. Software functionaliteit en MDA

3.1. Inleiding

In het vorige hoofdstuk is ingegaan op het MDA-raamwerk. In dit hoofdstuk wordt ingegaan op het begrip software functionaliteit en hoe een oordeel kan worden gegeven over de specificeerbaarheid hiervan op het PIM-abstractieniveau. Aangezien functionaliteit een abstract begrip is wordt functionaliteit onderverdeeld in verschillende typen functionaliteit. Zodoende kan er per type worden onderzocht in hoeverre dit type te specificeren is op het PIM-abstractieniveau en vervolgens getransformeerd kan worden naar het PSM-abstractieniveau. Voor het onderzoek naar de specificatie van functionaliteit op het PIM-abstractieniveau en de transformatie naar het PSM worden hypothesen opgesteld. Deze hypothesen worden getoetst door middel van de praktijk implementatie van het MDA-raamwerk van Compuware. De onderverdeling van (software) functionaliteit wordt beschreven in paragraaf 3.2. Per onderscheiden type functionaliteit worden hypothesen opgesteld, die worden beschreven in paragraaf 3.3. De toetsing van de hypothesen wordt beschreven in paragraaf 3.4 met behulp van Compuware's OptimalJ [B10]. Het hoofdstuk wordt afgesloten met een conclusie in paragraaf 3.5.

3.2. Onderverdeling software functionaliteit

Van Dale [D9] geeft de volgende omschrijving van functionaliteit:

func· ti· o· na· li· teit (de ~ (v.))
1 het functioneel zijn => nut
2 [psych.] wijze van reageren onder het aspect van primair of secundair

Het woord "nut" wordt vervolgens door van Dale omschreven als:

nut (het ~)
1 omstandigheid of eigenschap waardoor iets tot een doel kan dienen of voordeel kan opleveren => baat, functionaliteit, utiliteit, zin

Functionaliteit van een software systeem is dus één of meer eigenschappen van het software systeem die tot een doel kunnen dienen of voordeel kunnen opleveren.

Om te beoordelen in welke mate software functionaliteit specificeerbaar is op het PIM-abstractieniveau, wordt functionaliteit verdeeld in typen. Aangezien het niet het streven van het onderzoek is om alle mogelijke soorten functionaliteit te onderscheiden, is er een categorisatie gemaakt van de voornaamste typen van J2EE software functionaliteit. Om functionaliteit te classificeren, dient het type applicatie bekend te zijn, zodat er een zinnige classificatie gemaakt kan worden. Het type systeem is in dit geval een multitier web-based applicatie, gericht op het J2EE raamwerk.

De onderscheiden typen zijn bepaald aan de hand van contact met deskundigen binnen Ordina, literatuuronderzoek en door het evalueren van requirements en functionaliteit van verscheidene J2EE gebaseerde oplossingen en deze vervolgens te groeperen. Aan de hand van de onderscheiden typen functionaliteit kan een uitspraak worden gedaan over de specificeerbaarheid per type functionaliteit op het PIM-abstractieniveau. Naar aanleiding van deze diverse resultaten kan worden geconcludeerd in hoeverre functionaliteit in totaliteit (gericht op J2EE systemen) specificeerbaar is op het PIM-abstractieniveau van het MDA-raamwerk.

De volgende typen functionaliteit zijn onderscheiden:
- Business logic / rules
- Gegevensopslag / -verwerking

- Communicatie / gegevensuitwisseling

De onderscheiden typen worden in de volgende paragrafen verder behandeld.

3.2.1. Business logic / rules

Hieronder vallen zowel de statische als dynamische bedrijfsaspecten, die door business rules worden beschreven. Een business rule is een statement dat bepaalde bedrijfsaspecten vastlegt. Door de Business Rules Group [B1] worden drie typen onderscheiden, namelijk:

- *Structural assertion*
  Een statement dat de statische structuur van een bedrijfsaspect vastlegt, zoals "een factuur bestaat uit persoonsgegevens en productinformatie".
- *Action assertion*
  Een statement dat de constraints/voorwaarden van een organisatorische actie uitdrukt. *Action assertions* kunnen in autorisatie, condities en integriteit constraints worden onderverdeeld. Een autorisatie definieert of een bepaalde actie door een gebruiker mag worden ondernomen. Een conditie bepaalt of en welke business rule van toepassing is, als de conditie waar is, zoals "bevat de factuur een factuurnummer?". Een integriteit constraint specificeert een conditie die waar moet zijn, zoals "iedere factuur moet een factuurnummer bevatten".
- *Derivation*
  Een statement dat is afgeleid van andere kennis binnen het bedrijf. *Derivations* kunnen in wiskundige calculaties en afleidingen/gevolgtrekkingen worden onderverdeeld. Een wiskundige calculatie levert een feit op volgens een bepaald wiskundig algoritme, zoals "uurloon maal aantal uur". Een afleiding levert een feit op volgens logische inductie of deductie, zoals "het feit dat het uurloon op de urenregistratie in overeenkomst is met het uurloon van een persoon". Het opleveren van een feit resulteert in een toestandsverandering.

3.2.2. Gegevensopslag / –verwerking

Dit type dekt de functionaliteit van een software systeem die te maken heeft met het persistent maken van gegevens en de bewerking en verwerking van deze gegevens. Onder gegevens wordt de informatie verstaan die relevant is voor het bedrijfsproces, bijvoorbeeld persoonsgegevens, factuurinformatie en productgegevens. Onder bewerking en verwerking van gegevens vallen de acties aanmaken, lezen, wijzigen en verwijderen.

Het verwerken van gegevens gebeurt veelal door interactieloze programma's, zoals batch processen. Vandaar dat het type functionaliteit "interactieloze programma's" is ingedeeld onder deze soort functionaliteit. Interactieloze programma's zijn typisch programma's die een verzameling van taken uitvoeren, waarbij geen interactie met een mens vereist is. Een voorbeeld van een dergelijk proces is het verzamelen van rekeningoverzichten die naar diverse klanten worden verstuurd.

3.2.3. Communicatie / gegevensuitwisseling

Dit type is in de volgende twee groepen te verdelen:

- Communicatie tussen componenten/lagen van een systeem en tussen systemen onderling
  Gegevensuitwisseling binnen hetzelfde systeem speelt een rol zodra het software systeem is opgebouwd uit meerdere lagen, wat bij een J2EE systeem altijd het geval is. Zonder de communicatie tussen deze lagen is het systeem niet bruikbaar.

  Met de opkomst van Service Oriented Architecture (SOA) speelt de mogelijkheid tot het uitwisselen van gegevens tussen software systemen onderling een grote rol. Een typisch voorbeeld is het gebruik van een web service binnen een software systeem.
- Communicatie / interactie tussen gebruiker en systeem
  In een typisch J2EE systeem staat de interactie tussen gebruiker en systeem centraal. Er bestaat geen bruikbaar J2EE systeem zonder de mogelijkheid tot interactie met een gebruiker. Onder communicatie wordt het uitwisselen van informatie bestaan. Zowel het invoeren van gegevens

door een gebruiker als het presenteren van gegevens aan een gebruiker vallen hieronder. Het opgeven van een afleveradres van een bepaalde bestelling van een gebruiker is een voorbeeld van invoer van gegevens door een gebruiker. Het genereren van overzichten, zoals de bestelgeschiedenis van een gebruiker, is een voorbeeld van het presenteren van gegevens.

3.3. Hypothesen

Aan de hand van de in de vorige paragraaf besproken onderscheiden typen functionaliteit worden in deze paragraaf hypothesen opgesteld over de specificeerbaarheid ervan op het PIM-abstractieniveau. De volgende paragrafen beschrijven de hypothesen per type software functionaliteit. In paragraaf 3.4 worden de hypothesen getoetst. Elke hypothese begint met een accolade.

3.3.1. Business logic / rules

{**H1**}De statische structuur van een bedrijfsaspect (*structural assertion*) wordt in het UML klassenmodel vastgelegd. {**H2**}Autorisatie wordt vastgelegd in het UML klassenmodel in combinatie met het gebruik van OCL [C13, D5] en/of Action Semantics [C10, C16]. {**H3**}Condities en integriteits constraints kunnen volledig worden vastgelegd met OCL. {**H4**}Wiskundige berekeningen en afleidingen (*derivations*) kunnen met OCL en/of Action Semantics worden gedefinieerd.

3.3.2. Gegevensopslag / -verwerking

{**H5**}De klassen in het UML klassenmodel, inclusief de associaties tussen de diverse klassen, specificeren de gegevens die worden opgeslagen. {**H6**}De verwerking van gegevens (aanmaken, lezen, wijzigen en verwijderen, CRUD) kan worden gerealiseerd door UML interactiemodellen in combinatie met Action Semantics. Het skelet van een functie voor bijvoorbeeld het wijzigen van persoonsgegevens wordt gegenereerd vanuit een UML model en de precieze invulling van deze functie wordt met behulp van Action Semantics gerealiseerd.
{**H7**}Interactieloze programma's kunnen worden gerealiseerd door het gebruik van Common Warehouse Metamodel [C6, D8]. CWM bevat o.a. de package Warehouse Process, waarmee de uitvoering van transformaties[1] kan worden gemodelleerd.

3.3.3. Communicatie / gegevensuitwisseling

{**H8**}De communicatie tussen gebruiker en systeem wordt met UML interactiemodellen gespecificeerd, in combinatie met Action Semantics, waarmee de dynamische aspecten van een systeem worden gedefinieerd. {**H9**}Voor de gegevensuitwisseling met externe systemen kan er gebruik worden gemaakt van de UML Profile EDOC (Enterprise Distributed Object Computing) [C17]. Met het EDOC *Profile Element* ECA (Enterprise Collaboration Architecture) [C19] kan op het PIM-abstractieniveau de communicatie tussen systemen worden gemodelleerd.
{**H10**}De koppeling tussen verschillende lagen binnen een systeem wordt gerealiseerd door de (automatische) generatie van (communicatie-) bridges.

3.4. Toetsing van hypothesen

Aan de hand van de in paragraaf 3.3 opgestelde hypothesen per onderscheiden type functionaliteit wordt in deze paragraaf getracht de hypothesen te toetsen met behulp van Compuware's OptimalJ Architecture Edition versie 3.1 [B10, D7]. OptimalJ is een J2EE ontwikkelomgeving die gebaseerd is op het MDA-raamwerk en maakt onderscheid in de volgende modellen:

- Domain model (PIM)

[1] Anders dan bij een transformatie binnen het MDA raamwerk, is een transformatie binnen de context van CWM het ontleden, transformeren of laden van data.

Anders dan de naam doet vermoeden komt het *domain model* niet overeen met het CIM, maar met het PIM zoals in het MDA-raamwerk is voorgeschreven. Binnen het *domain model* wordt onderscheid gemaakt tussen *domain class model* en *domain service model*. Het domain class model komt overeen met het UML klassenmodel. Het *domain service model* dient voor het definiëren van dynamische informatie, zoals informatie die beschikbaar blijft tijdens een sessie.

- Application model (PSM)
  Het *application model* hanteert de J2EE implementatie van het *domain model* en kan bestaan uit DBMS, EJB en Web modellen.
- Integration model (PSM)
  Het *integration model* levert interfaces naar externe of bestaande systemen aan de hand van interface definities als WSDL, CORBA IDL en JCA COBOL.
- Code model
  Het *code model* bestaat uit de Java broncode van het software systeem.

Vooraf was het de bedoeling om de hypothesen te toetsen aan de hand van het uitwerken van een case, zoals in het plan van aanpak staat vermeld, zie bijlage A. De uitwerking van de case zou dan leiden tot het kunnen bewijzen of verwerpen van de opgestelde hypothesen. Tijdens het verkennen van de toetsingsmogelijkheden van de hypothesen is gebleken dat het merendeel van de hypothesen momenteel niet te toetsen is met OptimalJ. Aangezien het uitwerken van de case niets toevoegt aan het onderzoek met betrekking tot het toetsen van de hypothesen, is er voor gekozen de case niet verder uit te werken. De opgestelde case, die de typen software functionaliteit afdekt, is wel in bijlage C opgenomen, zodat er in toekomstig onderzoek gebruik gemaakt kan gaan worden. Hieronder volgt een kort overzicht van de toetsing van de opgestelde hypothesen.

De hypothesen {**H3**}, {**H4**}, {**H6**}, {**H7**}, {**H8**} en {**H9**} zijn momenteel niet te toetsen. Dit komt doordat OCL, Action Semantics, CWM en UML Profiles niet worden ondersteund door OptimalJ. Bovendien wordt uitsluitend het UML klassenmodel als input van transformaties ondersteund en niet de UML interactiediagrammen.

De hypothesen {**H1**}, {**H5**} en {**H10**} zijn volledig toetsbaar en zijn correct.
OptimalJ ondersteunt uitsluitend het UML klassenmodel als input van transformaties. De attributen van een klasse en de afhankelijkheden tussen de diverse klassen weerspiegelen de statische structuur van een bedrijfsaspect.
Het klassenmodel wordt getransformeerd naar een relationeel dataschema op het PSM niveau waaruit de SQL statements voor het creëren van de tabellen van de database worden gegenereerd.
De verschillende lagen van het te genereren software systeem worden door OptimalJ automatisch tijdens de transformaties aan elkaar gekoppeld.

Hypothese {**H2**} is deels toetsbaar.
Door gebruik te maken van een autorisatieklasse in het klassenmodel, die diverse inloggegevens en privileges van een gebruiker bevat, kan autorisatie gedeeltelijk als een statische structuur van een bedrijfsaspect in het klassenmodel worden vastgelegd (iedere gebruiker heeft een inlogaccount). Het dynamische aspect van autorisatie (zoals het valideren van de correctheid van de ingevoerde gebruikersnaam en wachtwoord) kan niet worden vastgelegd in het UML klassenmodel. OCL en Action Semantics worden op dit moment niet door OptimalJ ondersteund, zodat deze hypothese niet volledig getoetst kan worden.

Ondanks dat het onderzoeken van de alternatieven die door OptimalJ worden geboden voor het definiëren van de onderscheiden typen software functionaliteit buiten de reikwijdte valt, is er tevens onderzoek naar gedaan naar alternatieve methoden. Per onderscheiden type software functionaliteit wordt hieronder kort omschreven hoe de betreffende type wel te definiëren is met OptimalJ.

Business logic / rules
OptimalJ biedt een voorziening voor het vastleggen van business rules door middel van de *business expression library*. Het subtype *derivations* van business rules in de vorm van operaties/expressies kan worden opgesteld aan de hand van diverse talen (zoals Java en *regular expressions*). Deze verzameling van expressies wordt vervolgens vertaald naar één Java klasse met daarin de verscheidene expressies. Er moet handmatig code worden geschreven voor het aanroepen en verdere gebruik van de expressies. Deze oplossing van het vastleggen van business rules staat los van de abstractieniveaus, zoals voorgeschreven in het MDA-raamwerk. Ook wordt de mogelijkheid geboden om in de *domain service model* operaties te specificeren die enkel als skelet functioneren. De daadwerkelijke invulling van deze operaties geschiedt op het code abstractieniveau, wat tevens geldt voor operaties gespecificeerd in de *domain class model*. Het definiëren van autorisatie zal aanvullend op het statische aspect van autorisatie (in het *domain class model*) op dezelfde wijze geschieden. Zo kan er een functie worden gespecificeerd voor het valideren van de correctheid van de ingevoerde gebruikersnaam en wachtwoord, maar de daadwerkelijke invulling van deze operaties geschiedt op het code abstractieniveau. Het definiëren van condities en integriteits constraints geschiedt op dezelfde wijze als het definiëren van het subtype *derivations*, met het verschil dat het los staat van de *business expression library* en onderdeel uitmaakt van het PSM-abstractieniveau.

Gegevensopslag / -verwerking
De standaard CRUD acties worden door OptimalJ automatisch gegenereerd aan de hand van het *domain class model*. In het *domain service model* kunnen de toegangsrechten voor de CRUD acties door middel van een boolean waarde worden vastgelegd. Het realiseren van een interactieloos programma (batch job) zal op dit moment op de klassieke manier van ontwikkelen moeten worden gerealiseerd door middel van het schrijven van code.

Communicatie / gegevensuitwisseling
Onder de standaard CRUD acties die door OptimalJ automatisch worden gegenereerd aan de hand van het *domain class model* vallen ook het invoeren en presenteren van gegevens. Aanpassingen in deze standaard gegenereerde functies geschiedt op het code abstractieniveau.
Voor het gebruik van bestaande systemen kan er gebruik worden gemaakt van het *integration model*, waarmee interfaces worden geleverd naar externe of bestaande systemen aan de hand van interface definities als WSDL, CORBA IDL en JCA COBOL.

Het is gebleken MDA inclusief de relevante technieken, zoals OCL en Action Semantics, in de praktijk nog niet volwassen is. Dit is met name te wijten aan het ontbreken van standaards en de korte tijd dat MDA in ontwikkeling is. OptimalJ maakt gebruik van proprietary voorzieningen voor het definiëren van software functionaliteit, zoals de business expression library, zodat er van vendor-onafhankelijkheid nog geen sprake is. Wel wordt er gebruik gemaakt van XMI, zodat de UML modellen uitwisselbaar zijn met andere modelleer- en ontwikkelomgevingen. De door OptimalJ gebruikte transformaties voor het genereren van modellen zijn niet inzichtelijk, maar het is wel mogelijk om eigen transformaties te schrijven door middel van een OptimalJ-specifieke transformatiedefinitietaal. Het ontbreken van een standaard transformatiedefinitietaal in het MDA-raamwerk staat de open structuur van MDA in de weg. De code die uiteindelijk door OptimalJ wordt gegenereerd is aan te passen door het gebruik van *free-blocks*. In deze *free-blocks* kan functionaliteit worden gedefinieerd die momenteel niet op een hoger abstractieniveau te definiëren is.
De verschillen tussen een klassieke codegenerator en het MDA-raamwerk, zoals beschreven in paragraaf 2.5, zijn in praktijk nog niet geheel van toepassing.

4. Conclusies en aanbevelingen

Het onderzoek richt zich op de vraag in hoeverre software functionaliteit op het PIM-abstractieniveau van het MDA-raamwerk te definiëren is.

Ten eerste is op te merken dat het MDA-raamwerk nog volop in ontwikkeling is. Dit komt met name tot uiting in de mate waarin het MDA-raamwerk in de praktijk is geïmplementeerd. De mogelijkheden uit de theorie zijn momenteel niet geheel realiseerbaar in de praktijk. De bevindingen over de specificeerbaarheid van software functionaliteit op het PIM-abstractieniveau zijn gemaakt met behulp van Compuware's OptimalJ. Vanwege de beperkte onderzoeksperiode is er gekozen om slechts gebruik te maken van één MDA ontwikkelomgeving, namelijk OptimalJ. Het oordelen over de specificeerbaarheid van software functionaliteit op het PIM-abstractieniveau geeft hierdoor wellicht een vertekenend beeld van de mogelijkheden in de praktijk door het gebruik van andere ontwikkelomgevingen die gebaseerd zijn op het MDA-raamwerk. Desondanks is de verwachting dat met andere ontwikkelomgevingen vergelijkbare of zelfs mindere resultaten zullen worden behaald. De reden hiervoor is dat OptimalJ één van de best ontwikkelde MDA ontwikkelomgevingen is.

Ten tweede is in de praktijk gebleken dat J2EE software functionaliteit (nog) niet in zijn totaliteit gedefinieerd kan worden op het PIM-abstractieniveau. Meer dan de helft van de hypothesen kunnen momenteel niet worden getoetst met behulp van OptimalJ, aangezien deze MDA ontwikkelomgeving een aantal technieken niet ondersteunt. Voor het specificeren van het merendeel van de onderscheiden typen software functionaliteit biedt OptimalJ specifieke voorzieningen, die geen standaard binnen het MDA-raamwerk zijn. Buiten de statische structuur van een bedrijfsaspect (onderdeel van het type software functionaliteit business logic) worden business rules vastgelegd door middel van de *business expression library*, wat een OptimalJ-specifieke voorziening is voor het beheren van business rules. Ook kan er gebruik worden gemaakt van het *integration model*, waarmee interfaces worden geleverd naar externe of bestaande systemen aan de hand van interface definities.

De *business expression library* bestaat naast de abstractieniveaus PIM en PSM. Het gebruik van de business rules uit deze *business expression library* moet handmatig geprogrammeerd worden.
OptimalJ biedt geen voorziening waarbij de dynamische aspecten van een software systeem gespecificeerd kunnen worden anders dan door het schrijven van code. Condities en integriteits constraints worden op het PSM abstractieniveau vastgelegd, in tegenstelling tot het PIM-abstractieniveau.

Het is de verwachting dat de komende jaren de volwassenheid van MDA in de praktijk een vlucht gaat nemen. Het is aan te bevelen om de hypothesen die zijn opgesteld in de toekomst te herhalen. Met de komst van UML 2.0 wordt er veel functionaliteit toegevoegd met betrekking tot het dynamische aspect van het modelleren van software systemen en speelt OCL een grotere rol in UML. Zodra UML 2.0 wordt ondersteund door ontwikkelomgevingen die op MDA gebaseerd zijn zullen meer hypothesen getoetst kunnen gaan worden.

Wat interessant is voor toekomstig onderzoek is de (automatische) transformatie van het CIM naar het PIM en vice versa. Met de komst van UML 2.0 wordt dit aspect beter door de techniek ondersteund.

5. Evaluatie

In dit hoofdstuk wordt de afgelopen afstudeerperiode geëvalueerd. Hierbij wordt ingegaan op de afstudeerperiode in zijn algemeen, de planning en de punten die goed en minder goed zijn aangepakt.

Bij aanvang van de afstudeerperiode was ik nauwelijks bekend met het MDA-raamwerk. Door de vele technieken die binnen het MDA-raamwerk gebruikt worden was het een uitdaging om me te verdiepen in dit onderwerp. Het raamwerk is complex door de verscheidenheid aan technieken die gebruikt kunnen worden en de continue ontwikkeling van het raamwerk. Het is een interessante ontwikkeling in het software engineering vakgebied, aangezien een dergelijke aanpak het vakgebied drastisch kan veranderen.

Het onderzoek heeft de volgende resultaten opgeleverd:
- Kennis vergaard van het MDA-raamwerk
- Omschrijving van het MDA-raamwerk
- Typen software functionaliteit onderscheiden
- Hypothesen over specificeerbaarheid software functionaliteit op PIM-abstractieniveau
- Toetsing van hypothesen
- Bevindingen van implementatie van MDA in praktijk
- Conclusies en aanbevelingen

Een aantal verbeterpunten/constateringen:
- Het onderscheiden van typen software functionaliteit was niet onderkend in de planning
- Typen software functionaliteit sneller bepalen
- Meer tijd besteed aan literatuur dan gepland
- Hypothesen opstellen kostte meer tijd dan gepland
- Opstellen van case was niet nodig; toetsen van hypothesen kon ook zonder
- Planning niet correct, de onderzoeksperiode was korter
- Feedback vanuit de Universiteit van Amsterdam

Reflectie onderzoeksaanpak
Ten eerste was de vooraf opgestelde planning niet correct. Er was uitgegaan van een totale onderzoeksperiode van 13 weken, maar deze was in feite twee weken korter. Bovendien was de activiteit van het onderkennen van typen software functionaliteit niet onderkend in de planning. Het was beter geweest als de deadline voor het inleveren van de scriptie bij aanvang van de afstudeerperiode bekend was gemaakt om zodoende een beter afgestemde planning te kunnen maken.

In het plan van aanpak staat vermeld dat per type functionaliteit een case wordt opgesteld om de bijbehorende hypothese te kunnen toetsen, maar tijdens de afstudeerperiode is besloten om de onderscheiden typen software functionaliteit onder te brengen in één case. Tijdens de afstudeerperiode is besloten om de opgestelde case niet uit te werken, aangezien daarmee de hypothesen niet getoetst konden worden. Dat het uitwerken van een opgestelde case niet zou leiden tot het kunnen toetsen van de hypothesen was tijdens het opstellen van het plan van aanpak niet te voorzien.
In het plan van aanpak is ook sprake van het uitwerken van de opgestelde case in theorie. Ook hiervan is afgeweken aangezien de theoretische uitwerking al naar voren komt in de opgestelde hypothesen.
Buiten bovenstaande aanmerkingen had ik achteraf gezien het onderzoek op dezelfde manier aangepakt. Wel zal ik in het vervolg data vastleggen wanneer ik van plan ben een versie van de scriptie op te leveren. Zodoende wordt er iteratief aan de scriptie gewerkt.

Hieronder volgt per aspect van de afstudeerperiode een beoordeling.

Kwaliteit van het onderzoeksresultaat - 7
De conclusies zijn getrokken aan de hand van bevindingen van één MDA ontwikkelomgeving, wat bij aanvang van de afstudeerperiode was bepaald. Dit geeft wellicht een vertekenend beeld van de

mogelijkheden in de praktijk door het gebruik van andere ontwikkelomgevingen die gebaseerd zijn op het MDA-raamwerk. De hypothesen waren voor een groot deel niet te toetsen aangezien de technieken niet werden ondersteund. De kwaliteit van het onderzoek is gevoelsmatig hoger als het merendeel van de hypothesen wel getoetst kon worden.

Kwaliteit van de scriptie - 8
De onderwerpen die aan bod kwamen zijn goed en helder omschreven. Er is een duidelijk inzicht verschaft in het probleemgebied. De scriptie is goed opgebouwd en gestructureerd en het is prima leesbaar.

Moeilijkheidsgraad van de onderzoeksvraag - 9
Het onderwerp van het afstudeeronderzoek is redelijk complex. Het MDA-raamwerk is complex door de verscheidenheid aan technieken die gebruikt kunnen worden en de continue ontwikkeling van het raamwerk. Doordat het raamwerk in ontwikkeling is en er voor een aantal onderdelen (zoals de taal voor transformatiedefinities) geen standaarden zijn, is het concretiseren van de theorie niet altijd eenvoudig. Bovendien speelt de complexiteit van het onderwerp een grotere rol bij de korte duur van de onderzoeksperiode.

Relevantie van de vakken uit de Master Software Engineering voor de uitvoering van dit project - 9
Het afstudeeronderzoek sluit erg goed aan bij de Master Software Engineering. Het MDA-raamwerk is ten eerste een interessante ontwikkeling in het software engineering vakgebied, aangezien een dergelijke aanpak het vakgebied drastisch kan veranderen. De vakken waren inhoudelijk niet erg relevant voor de uitvoering van het afstudeeronderzoek. De technieken binnen het MDA-raamwerk zijn nauwelijks ter sprake gekomen. Software Architecture heeft de meeste relevantie met de uitvoering, vanwege het gebruik van UML in beperkte mate. De vakken zijn wel erg relevant bij het onderwerp van het afstudeeronderzoek. Hieronder wordt per vak de relevantie met het afstudeeronderzoek aangegeven:

- Software Construction; Het ontwikkelen van software zal bij volledig gebruik van het MDA-raamwerk behoorlijk gaan veranderen. In plaats van het traditioneel programmeren wordt er binnen het MDA-raamwerk gebruik gemaakt van modelleren.
- Software Evolution; De evolutie van software zal bij gebruik van MDA een meer natuurlijk proces worden. Bij het wijzigen van de onderliggende techniek of programmeertaal, hoeft in de ideale situatie enkel de transformatiedefinities te worden herschreven (of beter: deze zijn al geschreven door een derde partij), zodat de het gehele systeem opnieuw gegenereerd kan worden zonder aanpassingen in de modellen.
- Software Architecture; De term architectuur in de naam Model Driven Architecture is niet geheel op haar plaats. Het vakgebied software architectuur is omvangrijker dan het MDA-raamwerk doet vermoeden. Wel komen de componenten van een systeem en de onderlinge afhankelijkheden tot uiting in het raamwerk, wat ook een onderdeel is van architectuur. Maar een architectuur van een systeem is ook bedoeld om erover te kunnen praten met diverse stakeholders met verschillende achtergronden. Daarvoor is het MDA-raamwerk momenteel niet geschikt, omdat UML niet een taal is die door iedere stakeholder wordt begrepen.
- Software Process; MDA is niet afhankelijk van een bepaalt software process. Bij Extreme Programming wordt er in dit geval niet extreem geprogrammeerd, maar extreem gemodelleerd. Binnen RUP speelt UML een grote rol. De UML modellen kunnen dan worden gebruikt voor het generen van PSM(s) en code.
- Requirements Engineering; Op het moment speelt requirements engineering geen rol binnen het MDA-raamwerk. In de toekomst is het wellicht mogelijk om het CIM bij het automatische transformatieproces te betrekken. Requirements engineers kunnen gebruik gaan maken van het CIM.
- Software Testen; Het testen geschiedt op de traditionele wijze.

Literatuurlijst

Boeken

[A1] Kleppe, A., Warmer, J., en Bast, W., "MDA Explained: The Model Driven Architecture™: Practise and Promise", Addison-Wesley - Pearson Education, Inc. 2003, ISBN 0-321-19442-X

[A2] Frankel, D. S., "Model Driven Architecture™: Applying MDA™ to Enterprise Computing", Wiley Publishing, Inc. 2003, ISBN 0-471-31920-1

[A3] Mellor, S. J., en Balcer, M. J., "Executable UML: A Foundation for Model-Driven Architecture", Addison-Wesley 2002, ISBN 0-201-74804-5

[A4] Kleppe, A. en Warmer, J,. "Praktisch UML", Addison Wesley 1999, ISBN 90-6789-937-2

Artikels en presentaties

[B1] MODA-TEL Consortuim, "Assessment of the Model Driven Technologies – Foundations and Key Technologies", 2002 (http://www.modatel.org/~Modatel/pub/deliverables/D2.1-final.pdf)

[B2] Sims Associates, "MDA – The real value", 2002 (http://www.omg.org/mda/mda_files/OMG-Information-Day-Sims_01-01.pdf)

[B3] DSouza, D., "Model-Driven Architecture and Integration: Opportunities and Challenges", Version 1.1, February 2001 (http://www.catalysis.org/publications/papers/2001-mda-reqs-desmond-6.pdf)

[B4] Hendryx & Associates, "Integrating Computation Independent Business Modeling Languages into the MDA with UML 2", 17 januari 2003, document ad/03-01-32 (http://www.omg.org/cgi-bin/doc?ad/03-01-32)

[B5] Hendryx & Associates, "Response to the OMG BRWG Response to the OMG BRWG - Business Rules in Models RFI", 31 oktober 2002 (http://neptune.irit.fr/Biblio/02-11-02.pdf)

[B6] Gardner et al, "A review of OMG MOF 2.0 Query / Views / Transformations submissions and recommendations toward the final standard", 21 juli 2003 (http://www.omg.org/cgi-bin/apps/do_doc?ad/03-08-02)

[B7] The Business Rules Group, "Defining business rules – what are they really?", revision 1.3, juli 2000 (http://www.businessrulesgroup.org/first_paper/BRG-whatisBR_3ed.pdf)

[B8] Ahn, G., Shin, M.E., "Role-Based Authorization Constraints Specification Using Object Constraint Language", Tenth IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises, 20 - 22 juni 2001, page 157 – 163 (http://www.sis.uncc.edu/~gahn/papers/OCL.pdf)

[B9] Griffith University, School of Computing and Information Technology, "Business modeling: UML vs. IDEF" (http://www.cit.gu.edu.au/~noran/Docs/UMLvsIDEF.pdf)

[B10] King's College London, University of York, "An evaluation of Compuware OptimalJ Professional Edition as an MDA tool", September 2003

Standaards en specificaties

[C1] The Object Management Group, "MDA Guide Version 1.0.1" Document Number: omg/2003 -06-01 (http://www.omg.org/docs/omg/03-06-01.pdf) http://www.omg.org/cgi-bin/doc?mda-guide

[C2] The Object Management Group, "MDA – A technical perspective", document ormsc/2001 -07-01 (http://www.omg.org/cgi-bin/doc?ormsc/2001-07-01)

[C3] The Object Management Group, OMG MDA, http://www.omg.org/mda

[C4] The Object Management Group, "Unified Modeling Language, v1.5", document formal/03 -03-01 (http://www.omg.org/cgi-bin/doc?formal/03-03-01)

[C5] The Object Management Group, "Meta Object Facility, v1.4", document formal/2002 -04-03 (http://www.omg.org/cgi-bin/doc?formal/2002-04-03)

[C6] The Object Management Group, "Common Warehouse Metamodel Specification, v1.1", document formal/2003-03-02 (http://www.omg.org/cgi-bin/doc?formal/03-03-02)

[C7] The Object Management Group, "XML Metadata Interchange, v2.0", document formal/03 -05-02 (http://www.omg.org/cgi-bin/doc?formal/2003-05-02)

[C8] The Object Management Group, "MOF 2.0 Query / Views / Tr ansformations RFP", document ad/02-04-10 (http://www.omg.org/cgi-bin/apps/do_doc?ad/02-04-10)

[C9] CBOP, DSTC, IBM, "MOF Query / Views / Transformations, Second revised submission", 12 Januari 2004, document ad/2004-01-06 (http://www.dstc.edu.au/pegamento/publications/ad-04-01-06.pdf)

[C10] The Object Management Group, "UML 1.4 with Action Semantics, Final Adopted Specification", document ptc/02 -01-09 (http://www.omg.org/cgi-bin/apps/do_doc?ptc/02-01-09)

[C11] The Object Management Group, "UML 2.0 Infrastructure RFP" document ad/2000 -09-01 (http://www.omg.org/cgi-bin/apps/do_doc?ad/00-09-01)

[C12] The Object Management Group, "UML 2.0 Superstructure RFP", document ad/00 -09-02 (http://www.omg.org/cgi-bin/apps/do_doc?ad/00-09-02)

[C13] The Object Management Group, "UML 2.0 OCL RFP", document ad/00 -09-03 (http://www.omg.org/cgi-bin/apps/do_doc?ad/00-09-03)

[C14] Data Access Technologies et al, "Response to the OMG UML 2.0 Infrastructure RFP – UML4MDA Infrastructure for UML 2.0", 6 januari 2003, document ad/2003 -01-10 (http://www.omg.org/cgi-bin/doc?ad/2003-01-10)

[C15] Sun Microsystems, "Java 2 Platform Enterprise Edition Specification, v1.4", 11 -24-2003 (http://java.sun.com/j2ee/j2ee-1_4-fr-spec.pdf)

[C16] Kennedy Carter, "UML ASL Reference Guide, ASL Language Level 2.5", (http://www.kc.com/cgi-bin/download.cgi?action=ctn/CTN_06v2_5c.pdf)

[C17] The Object Management Group, "UML Profile for EDOC final adopted specification", document ptc/02-02-05 (http://www.omg.org/cgi-bin/apps/do_doc?ptc/02-02-05)

[C18] The Object Management Group, "UML for EAI Final adopted specification", document ptc/02-02-02 (http://www.omg.org/cgi-bin/apps/do_doc?ptc/02-02-02)

[C19] The Object Management Group, "Enterprise Collaboration Architecture (ECA), v1.0", document formal/04-02-01 (http://www.omg.org/cgi-bin/apps/do_doc?formal/04-02-01)

[C20] Tracy Gardner et al, "Review of OMG MOF 2.0 Query/Views/Transformations Submissions & Recommendations towards Final Std", document ad/03-08-02 (http://www.omg.org/cgi-bin/apps/do_doc?ad/03-08-02)

Webpagina's

[D1] http://www.omg.org/mda

[D2] http://www.bptrends.com/

[D3] http://www.martinfowler.com/bliki/

[D4] http://www.sdmagazine.com/

[D5] http://www.klassen.nl

[D6] http://www.middleware-company.com/

[D7] http://www.optimalj.com

[D8] http://www.cwmforum.org/

[D9] http://www.vandale.nl

Bijlage A: Plan van aanpak

Inleiding

Dit is het plan van aanpak van de afstudeerperiode van Wilfred Belo bij Ordina Clockwork betreffende het onderwerp "Model Driven Architecture". Het afstudeerproject zal 3 maanden duren, van 29 maart 2004 t/m 25 juni 2004. De student wordt begeleid door Jaap Branderhorst, ICT Consultant van Ordina Clockwork.
In dit plan van aanpak zijn de opdrachtomschrijving, de uit te voeren activiteiten, de planning, de projectorganisatie en de succes- en risicofactoren benoemd.

Projectomschrijving

Het onderzoek betreft de mogelijkheid voor het specificeren van functionaliteit van een J2EE systeem met behulp van Model Driven Architecture (MDA). Voor het daadwerkelijk in gebruik nemen van MDA in het J2EE ontwikkeltraject van Ordina, moet onderzocht worden of MDA in de praktijk wel bruikbaar is. De verwachting is dat de onderliggende structuur/techniek van een J2EE systeem op relatief eenvoudige wijze te realiseren is met MDA. Over het specificeren van functionaliteit van een systeem met behulp van MDA is meer onduidelijkheid.
Het aspect van het specificeren van functionaliteit van een systeem met behulp van MDA zal onderzocht moeten worden.

Dit leidt tot de volgende onderzoeksvraag:

- In hoeverre is het mogelijk om functionaliteit van een systeem op het hoogste abstractieniveau van MDA (het Platform Independent Model) te definiëren, zodanig dat het mogelijk is deze functionaliteit uiteindelijk te transformeren naar een werkend J2EE software systeem?

Als er binnen de stageperiode voldoende ruimte is worden tevens de volgende punten onderzocht:

- Welke voordelen biedt het specificeren van functionaliteit met behulp van MDA ten opzicht van het handmatige programmeerwerk?
- Hoe is de ondersteuning van verscheidene MDA-tools voor het specificeren van functionaliteit van een J2EE applicatie?

Verwachte resultaten

De volgende resultaten worden naar verwachting opgeleverd:

- Scriptie, bestaande uit:
    - Hypothesen
    - Cases
    - Resultaten uit theorie
    - Resultaten uit praktijk
    - Conclusie en aanbevelingen
- Praktijkuitwerkingen

Om de mogelijkheid te onderzoeken of functionaliteit definieerbaar is in het PIM-model, zal functionaliteit eerst verdeeld moeten worden in verschillende soorten functionaliteit. Aan de hand van de onderscheiden soorten functionaliteit zullen hypothesen over de definieerbaarheid van deze functionaliteit op het PIM-abstractieniveau worden opgesteld. Per onderscheiden categorie van functionaliteit zal een case worden opgesteld. Vanuit de theorie wordt bestudeerd of de betreffende case is te realiseren. Daarnaast wordt de case in de praktijk gebracht door het gebruik van OptimalJ.

De hypothesen zullen worden onderbouwd of verworpen aan de hand van zowel resultaten uit de theorie als praktijk. Naar aanleiding van deze resultaten kunnen er conclusies worden getrokken en aanbevelingen worden gedaan.
De praktijkuitwerkingen zullen bestaan uit modellen en/of code die naar aanleiding van het praktijkonderzoek zijn ontstaan.

Succes- en risicofactoren

De volgende tabel geeft een aantal onderscheiden risicofactoren weer in combinatie met het risico en de te nemen maatregelen:

Risicofactor	Kans	Schade	Risico	Maatregels
Geen beschikbaarheid van computerfaciliteiten op de werkplek	-	++	+	1) Ordina levert een laptop met voldoende capaciteit 2) Thuis gebruikmaken van de aanwezige computer
Onbekendheid van de technieken	-	++	+	1) Bestuderen van MDA m.b.t. functionaliteit 2) Bestuderen groter gebied van MDA
Beperkte inhoud van het onderzoek	-	++	+	
Korte onderzoeksperiode	+	++	++	1) Aantal hypothesen beperken 2) Cases eenvoudiger maken 3) Beperken tot theorie/praktijk uitvoering van case 4) Beperken tot hoofdzaken

Een toelichting van de risicofactoren volgt hieronder.

- Geen beschikbaarheid van computerfaciliteiten op de werkplek
  Het ontbreken of niet goed functioneren van deze faciliteiten zal het afstudeerproject een achterstand kunnen bezorgen. De kans dat een dergelijke storing optreedt is echter minimaal, maar de schade die het zal aanrichten aan het afstudeerproject is groot. Twee maatregelen die genomen kunnen worden zijn het leveren van een laptop met voldoende capaciteit door Ordina of het thuis gebruikmaken van de aanwezige computer.
- Onbekendheid van de technieken
  In dit afstudeerproject zal er gebruik gemaakt worden van technieken die niet of nauwelijks bekend zijn bij de student. MDA is een techniek waar de student nauwelijks mee bekend is. Het moet blijken hoe omvangrijk en complex deze techniek is.
  De kans dat een dergelijke storing optreedt is klein. De verwachting is dat de student door het volgen van zijn opleiding genoeg bagage heeft meegekregen om hier niet al te veel problemen mee te krijgen. Bovendien leunt MDA sterk op UML, waarmee de student ervaring heeft. De schade die een dergelijke storing aanricht is groot. Twee maatregelen die genomen kunnen worden zijn het bestuderen van de techniek MDA m.b.t. functionaliteit en het bestuderen van MDA in een groter gebied.
- Beperkte inhoud van het onderzoek
  Als blijkt dat het specificeren van functionaliteit van een J2EE software systeem, op een hoger abstractieniveau dan de source code, (nog) niet mogelijk is, zal het schade aanrichten aan de inhoud van het onderzoek. In hoeverre functionaliteit gespecificeerd kan worden op het PIM-abstractieniveau komt dan grotendeels te vervallen. Bovendien kunnen de twee optionele onderzoeksvragen niet goed beantwoord worden, omdat deze ook betrekking hebben op het specificeren van functionaliteit. De kans dat een dergelijk storing optreedt is klein.
- Korte onderzoeksperiode

Het afstudeerproject moet in een relatief kort tijdsbestek uitgevoerd worden, waardoor niet alle onderzoeksvragen aan bod kunnen komen. De kans dat de beperkte onderzoeksperiode een negatieve invloed heeft op het onderzoek is aanwezig en de schade die hierdoor wordt aangericht is groot. Een aantal maatregelen die genomen kunnen worden zijn het beperken van het aantal hypothesen (en bijbehorende soorten functionaliteit), het simplificeren van cases, het beperken tot de theoretische of praktische mogelijkheid van realisatie van een case en het beperken tot de hoofdzaken.
De twee optionele onderzoeksvragen zijn van toepassing als er binnen de stageperiode voldoende ruimte is om deze te beantwoorden.

Een aantal te onderscheiden succesfactoren zijn:
- De betrokkenheid van de bedrijfsbegeleider Jaap Branderhorst
- De betrokkenheid van de Universiteit van Amsterdam bij het afstudeerproject
- Duidelijke communicatie, o.a. over afspraken
- De uitdagende opdracht
- De interesse van de student in het afstudeeronderwerp
- Enthousiaste medewerkers

Uit te voeren activiteiten

De activiteiten die moeten worden uitgevoerd om de verwachte resultaten van het onderzoek te bereiken worden in deze paragraaf besproken. Per verwacht resultaat worden de activiteiten benoemd.

Scriptie - Hypothesen
- Literatuur en informatie op internet raadplegen om soorten functionaliteit te onderscheiden
- Contact met deskundige(n)/medewerker(s) voor het bepalen van soorten functionaliteit
- Vaststellen te beoordelen soorten functionaliteit
- Opstellen hypothesen aan de hand van onderscheiden soorten functionaliteit

Scriptie - Cases
- Opstellen cases aan de hand van onderkende soorten functionaliteit

Scriptie - Resultaten uit theorie
- Bestuderen van MDA met betrekking tot het specificeren van functionaliteit
- Literatuur en informatie op internet bestuderen op de mogelijkheid van theoretische realisatie van de cases

Scriptie - Resultaten uit praktijk
- Doornemen / analyseren OptimalJ
- Uitwerken cases in OptimalJ
- Literatuur en informatie op internet bestuderen op de mogelijkheid van praktische realisatie van de cases

Scriptie - Conclusie en aanbeveling
- Trekken van conclusie aan de hand van theoretische en praktische bevindingen naar aanleiding van de verschillende opgestelde cases
- Aanbevelingen geven voor het toekomstig gebruik van de resultaten van het project

Praktijkuitwerkingen
- Doornemen / analyseren OptimalJ
- Uitwerken betreffende case in OptimalJ

Methoden, technieken en standaarden

De techniek MDA neemt een centrale rol in binnen het afstudeerproject. De technieken waarvan MDA gebruikmaakt zullen voor zover van toepassing ook aan bod komen, zoals UML, OCL en MOF. Er zal gebruik worden gemaakt van het programma "OptimalJ" van Compuware om de concrete mogelijkheden op het gebied van het specificeren van functionaliteit met MDA uit te lichten. OptimalJ is één van de voornaamste en algemeen geaccepteerde MDA omgevingen voor het J2EE platform die veel facetten van de MDA standaard van de Object Management Group (OMG) ondersteunt. Als er tijdens het verloop van het afstudeerproject alsnog completere MDA omgevingen aan het licht komen en OptimalJ blijkt ontoereikend te zijn, zal bepaald worden of er wordt overgestapt.

Relaties met andere projecten

Binnen Ordina Clockwork vindt er een ander afstudeeronderzoek plaats over de toepasbaarheid van MDA binnen het ontwikkelproces van Ordina Clockwork. De twee onderzoeken zullen een minimale overlap hebben aangezien het ene onderzoek zich richt op andere facetten van MDA, zoals de toepasbaarheid binnen het Ordina Clockwork ontwikkeltraject.

Communicatie

De communicatie binnen het bedrijf zal voor het grootste gedeelte op een informele manier geschieden. De voortgang van het afstudeerproject wordt met de bedrijfsbegeleider besproken. De bedrijfsbegeleider zal ook commentaar leveren op de scriptie.

Kostenschatting

Aan de aanschaf van OptimalJ zijn licentiekosten verbonden, maar het is niet geheel duidelijk hoe hoog deze kosten zijn. Er is wel een evaluatie versie van 30 dagen beschikbaar.
Als er binnen de stageperiode voldoende ruimte is voor het onderzoeken naar de mogelijkheden van de verschillende MDA-tools, zullen deze ook aangeschaft moeten worden, tenzij er evaluatieversies beschikbaar zijn.

Planning

Bij deze planning is er uitgegaan van een werkweek van 5 dagen en een afstudeerperiode van 13 weken (29 maart 2004 t/m 25 juni 2004).

- 2 weken: bestuderen van MDA m.b.t. het specificeren van functionaliteit
- 1 week: opstellen hypothesen / MDA cases
- 6 weken: uitwerken MDA cases, bestaande uit:
    - Doornemen analyseren OptimalJ
    - Uitwerken MDA cases in theorie
    - Uitwerken MDA cases in praktijk
- 3 weken: rapportage / scriptie
- 1 week: uitloop

Bij deze planning is de uitvoering van de twee onderzoeksvragen, die worden uitgevoerd als er binnen de stageperiode voldoende ruimte is, niet meegenomen.

Bijlage B: MDA technologieën

Unified Modeling Language (UML)
UML [A4, C4] is een OMG standaard voor objectgeoriënteerd ontwerp en analyse. Het is een taal voor het specificeren, visualiseren, ontwerpen en documenteren van objecten binnen software systemen, maar ook voor overige systemen. UML is de meest gebruikte taal voor het modelleren en is de sleuteltechniek van MDA.

Object Constraint Language (OCL)
OCL is onderdeel van UML en is een taal waarmee het mogelijk is om expressies en constraints te beschrijven. OCL is declaratief, waar dus geen dynamisch gedrag kan worden gedefinieerd. Er worden twee typen constraints onderscheiden, namelijk pre- en post-conditions en invariants.

UML Action Semantics (AS)
De UML Action Semantics is een uitbreiding van UML waarmee een basis wordt gelegd voor de dynamische semantiek van UML. Er is geen concrete syntax gegeven in de standaard, waardoor er geen statements in een gestandaardiseerde manier kunnen worden geschreven. Met Action Semantics worden de dynamische aspecten van een applicatie gedefinieerd, dat wil zeggen de procedures die leiden tot toestandveranderingen van objecten.

Executable UML (xUML)
Executable UML [A3] is gedefinieerd als UML in combinatie met Action Semantics. De data wordt vastgelegd in het UML klassenmodel en de interactie met de UML toestandsdiagram. De algoritmes worden gedefinieerd met Action Semantics.

Meta Object Facility (MOF)
De MOF is een OMG standaard die de taal definieert voor het definiëren van modelleertalen. MOF is gedefinieerd met MOF zelf.

Common Warehouse Metamodel (CWM)
CWM [C6, D8] is een modelleertaal dat specifiek bedoelt is voor het modelleren van data warehouse applicaties. CWM is gebaseerd op MOF en kan worden beschouwd als een UML specifiek model voor data warehouses.

UML Profiles
Een UML Profile definieert een specifieke wijze waarop gebruik wordt gemaakt van UML. Een aantal UML Profiles zijn UML Profile for CORBA, UML Profile for Enterprise Application Integration (EAI) en UML Profile for Enterprise Distributed Object Computing (EDOC).

Query, Views and Transformations (QVT)
Een standaard in ontwikkeling en zal een onderdeel van de MOF gaan worden. QVT [C8, C9] definieert de manier waarop transformaties worden uitgevoerd tussen modellen, waarvan de talen waarin deze zijn geschreven gedefinieerd zijn met de constructies uit de MOF. Een query is een expressie die over een model wordt geëvalueerd, wat resulteert in één of meer instanties die zijn gedefinieerd in het bronmodel of door de query-taal [C20]. Een voorbeeld van een query over een UML model is: "Geef alle klassen terug die niet worden overgeërfd". Deze query zou resulteren in een collectie van instanties van de metaklasse "Klasse".

Een view is een model dat volledig is afgeleid van een ander model, het basismodel, en kan niet onafhankelijk van het basismodel worden gewijzigd. Views die wijzigbaar zijn hebben een transformatie nodig, zodat wijzigingen tevens worden doorgevoerd in het basismodel. Een view richt zich vaak op een onderdeel van het basismodel, specifiek voor een bepaalde taak of gebruiker. Views worden gegenereerd door middel van transformaties.

Een transformatie is het proces dat een doelmodel genereert vanuit een bronmodel, het basismodel. Transformaties kunnen *top-down*, *one-way* (unidirectioneel) of *two-way* (bidirectioneel) zijn. In een

top-down transformatie worden er geen wijzigingen aangebracht in het doelmodel, maar uitsluitend in het bronmodel. *One-way* transformaties voeren wijzigingen in het bronmodel door naar het doelmodel, maar doelmodel-specifieke informatie wordt niet overschreven. Tijdens *two-way* transformaties worden zowel wijzingen in het bronmodel als doelmodel respectievelijk doorgevoerd in het doelmodel en bronmodel.

XML Metadata Interchange (XMI)
XMI wordt gebruikt als een standaard mechanisme voor het uitwisselen van modellen op basis van XML.

Bijlage C: Case

Onderstaande case is fictief. Er wordt in de case geen rekening gehouden met specifieke kwaliteitseisen, zoals beveiliging, aangezien deze buiten de reikwijdte van het onderzoek vallen en om de case niet te complex te maken. In de case zijn de onderscheiden typen software functionaliteit verwerkt.

Een kleine bank wil haar klanten de mogelijkheid geven tot het regelen van bankzaken via het internet. De klanten/rekeninghouders loggen in via een webpagina met behulp van de door de bank verstrekte wachtwoorden en gebruikersnamen. Na succesvol inloggen wordt de gebruiker naar de welkomstpagina verwezen van waar de gebruiker verdere acties kan ondernemen. De volgende acties kunnen worden ondernomen door een klant:

- Inzien van transacties van de klant van de afgelopen twee weken
- Overschrijven van geld naar rekeninghouder van dezelfde bank aan de hand van rekeningnummer
- Opvragen huidige saldo
- Opvragen maximaal te lenen bedrag, dat wordt berekend met de formule: (inkomen * 0.4 / 2 + 120) * 14
- Wijzigen van personele gegevens, gebruikersnaam en wachtwoord
- Aanmaken van een nieuwe rekening, met een maximum van twee rekeningen per account
- Rekening opheffen, mits het saldo niet negatief is.

Als er een persoon op de website arriveert die een rekening wil openen, is er de mogelijkheid tot het registreren als klant via de website. De gebruiker dient persoonlijke gegevens in te voeren, waarna de gebruiker wordt doorverwezen naar de welkomstpagina. Er zijn een aantal voorwaarden waaraan een nieuwe klant moet voldoen:

- Leeftijd >= 18 jaar
- De persoon staat bekend als “goede betaler” bij externe instantie B, wat wordt gevalideerd door het doorgeven van de naam van de betreffende persoon aan de instantie
- Wachtwoord en gebruikersnaam zijn langer dan 5 karakters
- Naam, wachtwoord, gebruikersnaam, adres, geboortedatum, sofi-nummer en inkomen per maand zijn ingevoerd
- Telefoonnummer of emailadres is ingevoerd

Eén keer per dag krijgen de rekeningen die zijn opgeheven, met een positief saldo, het saldo nul. Alle gegevens, zoals klant- en rekeninginformatie, worden opgeslagen.