Scriptie:

Softwarekwaliteit van MDA tools

Afstudeerproject : Softwarekwaliteit van MDA tools
Datum : 31 augustus 2005

Auteur : Ing. E. de Groot
Emailadres : E.degroot@student.uva.nl
Afstudeerdocent : Prof. Dr. D.J.N. van Eijck

Afstudeerbedrijf : Centrum voor Wiskunde en Informatica
Begeleider : Dr. M.G.J. van den Brand

Universiteit van Amsterdam

Master Software Engineering aan de Universiteit van Amsterdam, FNWI

Voorwoord

Voor u ligt de scriptie van het afstudeerproject "Softwarekwaliteit van MDA tools". Dit afstudeerproject gaat over de softwarekwaliteit van Model Driven Architecture, in opdracht van het Centrum voor Wiskunde en Informatica. Hiermee wordt de opleiding master Software Engineering aan de Universiteit van Amsterdam afgesloten. Het afstudeerproject heeft drie maanden geduurd en door middel van deze scriptie wordt aangetoond dat aan de eindtermen van de master Software Engineering is voldaan.

Mijn dank gaat uit naar mijn ouders die het voor mij mogelijk hebben gemaakt deze opleiding te volgen en mijn vriendin welke een grote steun was tijdens de opleiding. Ook gaat mijn dank uit naar de bedrijven die hebben meegeholpen aan het onderzoek, dit zijn InfoProfs en Compuware. Compuware heeft mij de mogelijkheid geboden om hun softwarepakket te kunnen onderzoeken en daarvoor een licentie van een halfjaar afgegeven. Mijn hartelijke dank hiervoor.

Tot slot wil ik nog mijn begeleiders Dr. M.G.J. van den Brand en Prof. Dr. D.J.N. van Eijck bedanken voor hun begeleiding en alle docenten, van de Universiteit van Amsterdam, die deze opleiding hebben verzorgd.

Edwin de Groot
Master Software Engineering 2004-2005

Inhoudsopgave

Samenvatting

Deze scriptie gaat over de softwareontwikkelingmethode Model Driven Architecture [MDA], afgekort MDA. Vaak wordt MDA verward met MD(S)D, wat staat voor Model Driven (Software) Development [MDSD]. Er is echter een groot verschil tussen de verschillende softwareontwikkeling-methodes. MDA begint op een hoger abstractieniveau dan MDD, namelijk bij het modelleren van het business onafhankelijk model. Dit model beschrijft alle requirements, de omgeving en wat er van het systeem verwacht wordt. Vaak wordt dit model bij een MDA aanpak overgeslagen omdat de requirements al gemaakt zijn.
Het model waar MDA meestal mee begint is het platform onafhankelijk model. In dit model wordt namelijk nog geen architectuurbeslissingen en specifieke platforminformatie verwerkt. Middels een transformatie wordt het onafhankelijk model omgezet naar een platform afhankelijk model, waarbij de architectuurbeslissingen en platformspecificatie wel zijn verwerkt. Dit model is het startpunt van MDD, waarbij dus de toekomstige applicatie meteen in een platform afhankelijk model gemaakt wordt. Als alle platformspecifieke wijzigingen in het platform afhankelijk model zijn gemodelleerd kan vanuit dit model de code gegenereerd worden. Na mogelijk wat wijzigingen in de code te programmeren kan de code gecompileerd worden, waarna de applicatie klaar voor testen is. De transformaties tussen de modellen en code gebeuren door middel van patterns, welke beschrijven hoe de brontaal tot het doeltaal moet komen.

In dit onderzoek wordt gekeken naar de kwaliteit van MDA tools. Waarbij gekeken wordt naar hoe de tools omgaan met schaalbaarheid van modellen, wat de prestaties zijn en of het gewenste resultaat is behaald. Gezien de tijd is er uit verschillende MDA tools, zoals bijvoorbeeld: XDE [IBM], ArcStyler [IO], ArchitectureWare [OAW] en AndroMDA [AndroMDA], één tool gekozen. Er is gekozen voor Compuware's OptimalJ 3.3 Professional Edition, dit tool wordt in detail beschreven in deze scriptie en dient als uitgangspunt voor het onderzoek. Zo heeft OptimalJ diverse functionaliteiten, zoals: bescherming van de geschreven of gegenereerde code, ondersteuning voor versiebeheer systemen, het testen en debuggen van de applicatie en een applicatieserver.
Aan de hand van de definitie en uitleg van MDA moet vastgesteld worden of OptimalJ een MDA tool is. Hiervoor is er eerst gekeken of de verschillende aspecten van de voorgeschreven MDA standaards in de tool voorkomen. Hieruit is geconcludeerd dat OptimalJ wel dergelijk een MDA tool is.

Voor het onderzoek zijn er 3 casussen opgesteld, welke alle drie een opeenvolging van elkaar zijn. Dit zodat de schaalbaarheid en prestatie goed onderzocht kan worden. De casussen representeren een reeds ontwikkeld systeem van de IJmuider Reddingsbrigade[1] (IJRB). Na het uitvoeren van deze casussen zijn alle gegevens verzameld en in tabellen en grafieken verwerkt.
Vervolgens zijn er verschillende vragen ten behoeve van het onderzoek beantwoord met behulp van deze gegevens en de bevindingen van de onderzoeker. Dit resulteerde in de observatie dat de modellen in OptimalJ daadwerkelijk schaalbaar zijn en men met een beperkte hoeveelheid intern geheugen een redelijke applicatie kan genereren. Ook is er geconcludeerd dat de tool prima presteert en er een verband is met de grootte van het model en de transformatietijd. Als laatste is de kwaliteit van de code en het eindresultaat geëvalueerd. Hierbij enkele kritische opmerkingen: de logica wordt niet gegenereerd en de applicatie werkte niet 100% door een onbekende fout. Dit alles is in de eindconclusie samengebracht.

[1] http://www.ijrb.nl

1 Aanleiding

In de loop van de jaren hebben er verschillende veranderingen in de ontwikkeling van software plaatsgevonden. Eén van de grootste veranderingen is de overgang van de assembler naar de compiler. Met deze stap was niet iedereen het eens en veel programmeurs van die tijd zijn dan ook niet direct overgegaan op de nieuwe ontwikkelingstechniek. Tegenwoordig zien we dat verschijnsel nog steeds terug, we blijven kritisch bij het gebruik van andere technieken of methodes. Dit is geen verkeerde houding, omdat sommige technieken hetzelfde zijn als oude mislukte technieken maar dan in een nieuw jasje. Zodoende wordt er in Nederland en de rest van de wereld veel tijd besteed aan het onderzoeken van die nieuwe technieken en methoden.
Vandaag de dag zien we nog een grote verandering in de ontwikkeling van software, namelijk het model gedreven ontwikkelen van software. De hoofdgedachte hiervan is dat de ontwikkelaar modellen maakt om vervolgens vanuit deze een applicatie te genereren. Dit zou betekenen dat men niet meer uren hoeft te programmeren aan standaard oplossingen, maar deze worden voor je gegenereerd middels een model. Er zijn twee model gedreven methoden, namelijk: Model Driven Architecture [MDA] en Model Driven (Software) Development [MDSD]. Alle twee lijken ze erg op elkaar maar toch zijn er duidelijk verschillen waarneembaar.
Na het lezen van de scriptie van de student W. Belo [Belo 04] is de interesse gewekt om deze nieuwe methodes te onderzoeken. In dit onderzoek wordt er gekeken naar wat Model Driven Architecture en Model Driven (Software) Development is. Tevens wordt er een MDA tool onderzocht betreffende de softwarekwaliteit.

1.1 Probleemstelling

Het onderzoek betreft de kwaliteit van Model Driven Architecture (MDA) tools. MDA is een methode waarmee men software kan genereren door middel van (formele) modellen. Deze methode wordt in hoofdstuk 0 behandeld. Tegenwoordig zijn er verschillende MDA tools verkrijgbaar en in papers worden er allerlei kleine casussen behandeld. Om te kijken of deze tools ook grote modellen aan kunnen wordt bijvoorbeeld de schaalbaarheid onderzocht.
Gezien de beschikbare onderzoekstijd van drie maanden wordt er maar één MDA tool onderzocht. We hebben de keuze uit verschillende soorten MDA tools [Czarnecki 03], zoals onder andere: XDE [IBM], ArcStyler [IO], ArchitectureWare [OAW], AndroMDA [AndroMDA] en OptimalJ [Compuware]. Deze tools komen uit zowel de close als open source wereld. De keuze van het onderzochte MDA tool is OptimalJ van Compuware[2] geworden. Dit is tot stand gekomen doordat OptimalJ enorm wordt aangeprezen als een goed MDA tool en Compuware de mogelijkheid bood om gedurende de onderzoeksperiode over een licentie te beschikken. Daarbij komen er vanuit het bedrijfsleven wat negatieve klanken naar voren over het gebruikt van OptimalJ. Vanuit deze invalshoek wordt er gekeken in hoeverre OptimalJ daadwerkelijk een MDA tool is. Ook wordt er gekeken of deze verschillende grootte van modellen aankan en of de gegenereerde applicatie bruik-/werkbaar is. Een medestudent heeft, met behulp van de MDA tool ArcStyler, onderzocht welke problemen een ontwikkelaar zou tegen komen bij het gebruik van een MDA tool [Meijles 05]. Dit is een kleine opstap naar wat ik met dit onderzoek wil bereiken.

De probleemstelling is:

Hoe staat het met de kwaliteit van het MDA tool OptimalJ? In hoeverre is het mogelijk vanuit een groot UML model een applicatie te laten genereren? Wat is de kwaliteit van het eindresultaat en hoe staat dit tegenover de benodigde tijd om dit te realiseren?

Het onderzoek is onderverdeeld in de volgende onderdelen:

- Gebruiksvriendelijkheid van MDA tools;
- Prestatie van MDA tools, hard- of software grenzen;
- Hoeveelheid tijd er nodig is voor het genereren van een applicatie ten opzichte van het gewenste resultaat;
- In hoeverre de grootte van het UML model de werking van de tools benadeelt;
- Kwaliteit van de gegenereerde code, applicatie of raamwerk.

Voor dit onderzoek zijn 3 casussen opgesteld, welke zullen bijdragen aan het onderzoeken van de bovenstaande punten.

[2] http://www.compuware.com

1.2 Literatuurstudie

Voor het onderzoek is er gekeken naar verschillende artikelen met betrekking tot het onderzoeken van de softwarekwaliteit van Model Driven Architecture (MDA). Gezien het onderwerp MDA redelijk nieuw is zijn er nog relatief weinig papers over geschreven. In [Brand 04] wordt softwarekwaliteit in verband met o.a. MDA, codegeneratoren behandeld, maar meer in de context van het lesgeven daarin en wat voor toegevoegde waarde de kwaliteit van software heeft bij het ontwikkelen ervan. Er zijn ook papers te vinden welke alleen op het onderwerp MDA ingaan, daarbij kijken naar wat het inhoud en proberen een sluitende definitie vast te stellen. Ragnhild Kobro Runde en Ketil Stolen hebben dit in hun paper [Runde 03] voor de universiteit van Oslo proberen te doen. Zij merken op dat zelfs de woorden "model driven" niet eens goed gedefinieerd zijn. Met behulp van verschillende artikelen proberen zij een sluitende definitie te geven voor "model driven". Zodoende bespreken ze ook de specifieke eigenschappen van MDA en proberen hiervoor handvatten te geven waar de officiële documentatie [MDA] te kort aan doet. In mijn onderzoek is een soort gelijke opzet gebruikt. Met behulp van boeken, papers en documentatie, o.a. [Runde 03, Kleppe 03, Raistrick 04 en MDA], wordt er in hoofdstuk 2 besproken wat MDA inhoudt. Dit voornamelijk om dezelfde reden als [Runde 03], de ontoereikendheid van de officiële documentatie [MDA]. Het geeft geen duidelijk beeld wat MDA is en inhoudt, alleen paragraaf 2.1.2 van [MDA] geeft een duidelijk en concreet overzicht.

Vaak verward men Model Driven Architecture met Model Driven (Software) Development (MDD of MDSD). Papers en documentatie over MDD zijn minder te vinden dan over MDA. Dit heeft waarschijnlijk te maken met het feit dat MDA door de Object Management Group wordt ontwikkeld. Ook worden er veel papers over MDA door Compuware verspreidt, dit met betrekking tot hun product OptimalJ. Veelal van deze papers zijn white papers en lijken niet de gebruikelijke wetenschappelijke kritische blik te bevatten. Om deze reden zijn dit soort papers dan ook niet in dit onderzoek opgenomen en/of voor gebruikt. De papers die zijn gebruikt om een beter beeld op te bouwen over Model Driven (Software) Development zijn [Greenfield 03, Völter 04].

Uiteraard is er ook gekeken naar papers welke een soort gelijk onderzoek hebben uitgevoerd. Zoals eerder al vermeld is er genoeg te onderzoeken op het gebied van model gedreven software-ontwikkeling. Dit maakt ook dat dit soort profiling onderzoeken nog niet veelvuldig gedaan zijn. Enkele onderzoeken het wel gedaan hebben zijn [Belo 04, Witkop 03, Naijb 03, Mohag 04, Middleware_1 03, Middleware_2 03 en Middleware_3 03]. Deze papers lopen uit tot het uitvoeren van een casus met de onderzoeksvraag of MDA wat voor het bedrijf is tot de onderzoeksvraag of een tool bruikbaar is binnen een bedrijf.
De schrijvers van de paper [Naijb 03] zijn Selo & Warsun Naijb en schreven deze scriptie voor hun afstuderen aan de Agder Universiteit in Grimstad (Noorwegen) in samenwerking met Ericsson[3]. Zij schrijven over MDA en de integratie van legacy systemen. Hierbij wordt MDA uitvoerig besproken en een aantal MDA tool bekeken. Zij hebben een casus met een PIM model, waarbij ze in het onderzoek grondig kijken of dit werkelijk een PIM is. Ook in dit onderzoek wordt er gekeken of MDA wat is voor het bedrijf Ericsson. In [Mohag 04] wordt onderzocht hoe er met reverse engineering van de Erlang code een UML/XMI code eruit vertaald kan worden. Dit onderzoek ook samen met Ericsson gedaan en heeft een paar extra schrijvers Selo, Warsun Naijb, Jan Pettersen Nytun en Parastoo Mohagheghi. Een interessant onderzoek, waarbij gebruikt wordt gemaakt van een industriële casus, met de intentie om van bestaande modellen onafhankelijke modellen te maken. Dit is beschreven in [Naijb 03] als toekomstig werk, namelijk de PIM generator. Waarbij ze een vertaler willen ontwikkelen die Erlang of CORBA code willen vertalen naar XMI.
In de paper [Middleware_3 03] wordt de model gedreven J2EE ontwikkeling bekeken en daarbij twee aanpakken en tools met elkaar vergeleken. Deze twee aanpakken en tools zijn al reeds onderzocht en beschreven in de papers [Middleware_1 03 en Middleware_2 03] en worden nu met elkaar vergeleken. De aanpakken waar we het over hebben zijn MDA en Architected Rapid Application Development (ARAD) en de tools zijn Compuware's OptimalJ en IBM's Rational Rapid Developer tool. In de paper wordt er nog een kort overzicht gegeven wat MDD, ARAD en MDA is en worden deze aanpakken met elkaar vergeleken. In het hoofdstuk 4.4 daarna worden de methodes MDA en ARAD met elkaar vergeleken. Hieruit komen de volgende overeenkomsten:

- Beide methodes beginnen met een model dat een complete beschrijving geeft van de te genereren applicatie.
- In beide gevallen wordt het model gebruikt om de meeste applicatiecode te genereren, op basis van de gekozen technologie en platform.

[3] http://www.ericsson.com

- De codegeneratie gebeurt op basis van architecturale best practice.
- Op een gegeven moment kan het code generatieproces worden bestuurd.

Uit de paper [Middleware_3 03] komen ook de verschillende tussen ARAD en MDA, welke zijn:

- MDA is gebaseerd op een formele standaard, zoals bijvoorbeeld CORBA of UML. Bij ARAD is het gebruik van standaards niet gegarandeerd.
- MDA definieert speciaal drie lagen van modellen, het PIM, PSM en code model. Door de niet standaardisatie bij ARAD wil dit wel eens verschillen per tool.

De andere Middleware onderzoeken zoals [Middleware_1 03] waarin de productiviteit van MDA onderzocht wordt. Hierin wordt geconcludeerd dat men middels MDA 10 tot 20% sneller een applicatie kan ontwikkelen. Voor bedrijven en onderzoeken zoals deze is het interessant dit te lezen en verder te onderzoeken.

1.3 Onderzoeksverantwoording

Indien men dit onderzoek wil heronderzoeken zou het mogelijk kunnen zijn dat op een ander computersysteem er andere waarden naar voren kunnen komen. De computer waarop het onderzoek is verricht heeft de volgende specificaties: P4 2,5 GHz, 512MB, SATA 80 GB HD en een bussnelheid van 533 MHz. Op de computer is Windows Professional Edition en Norton Antivirus2004 geïnstalleerd. Er zijn zoveel mogelijk programma's uitgeschakeld om een zodanig groot mogelijk geheugen vrij te hebben. De geheugenwaarden zijn afgelezen met behulp van het programma "Memory Spie" van Eventcorder en het taakbeheer van Windows. Indien het geheugen tijdens een bewerking fluctueert, dan wordt het gemiddelde hiervan genomen.

Van het totale werkgeheugen van 490224 KB wordt 167076 KB gebruikt (34%) en heeft het wisselbestand een grootte van 236384 KB.

1.4 Leeswijzer

Hier krijgt u een korte beschrijving waar de verschillende onderdelen van het onderzoek te vinden zijn. Zo kunt u in hoofdstuk 2 lezen over Model Driven Architecture en de relatie die deze heeft met Model Driven (Software) Development. Hierna wordt in hoofdstuk 3 gekeken naar de MDA tool OptimalJ 3.3 Professional Edition van Compuware. Verschillende aspecten komen in dit hoofdstuk aan bod, waaronder de lay-out, de soorten modellen, de installatie-eisen, de functionaliteiten, MDA in OptimalJ en de schaalbaarheid van de modellen volgens Compuware.

De resultaten van de uitgevoerde casussen kunt u lezen in hoofdstuk 4, waarbij de uitwerking van de casussen terug te vinden zijn in bijlage A. Na het bestuderen van de gegevens, worden de bevindingen met betrekking tot het onderzoek en de subconclusies in hoofdstuk 5 beschreven. Als laatste wordt in hoofdstuk 6 de eindconclusie besproken. In bijlage B vind u het plan van aanpak dat voor dit onderzoek gebruikt is terug.

2 Model Driven Architecture

Model Driven Architecture (MDA) [MDA] is ontwikkeld door Object Management Group[4] (OMG) waarmee zij beweren dat ze vanuit modellen een applicatie kunnen genereren. OMG bestaat al sinds 1989 en probeert industriële richtlijnen en object management specificaties vast te stellen voor een algemeen framewerk voor applicatieontwikkeling [UML]. In dit hoofdstuk proberen we de definitie van MDA vast te stellen, de verschillen met Model Driven (Software) Development [MDSD] aan te wijzen en dieper op de verschillende onderdelen van model gedreven softwareontwikkeling in te gaan.

2.1 Definitie van MDA

Om MDA te begrijpen en te bepalen of een tool volgens deze methode werkt moet er eerst gekeken worden naar de definitie. Helaas zijn er verschillende definities van MDA te vinden en zijn deze niet al te duidelijk. Zodoende is er een literatuuronderzoek geweest betreffende een goede definitie. Met behulp van diversen papers [o.a. Belo 04, Gardner 02, Runde 03, Naijb 03], boeken [Kleppe 03, Raistrick 04] en documentatie/specificaties [MDA en IEEE 02] is er een duidelijke definitie en uitleg tot stand gekomen. De definities zijn als volgt:

Een *Model* is de beschrijving of specificatie van het systeem en zijn omgeving voor bepaalde bedoelingen. Modellen worden vaak gecombineerd zowel met tekeningen als met tekst. [MDA]

Model Driven is de aanpak van softwareontwikkeling welke de kracht van modellen volledig benut. Het geeft een betekenis aan modellen om het ontwerp, de constructie, de implementatie, de operaties, het onderhoud en het aanpassen enige richting te geven. [MDA]

Een *Architecture* is de fundamentele organisatie van het systeem met zijn componenten en hun onderlinge relaties. [IEEE 02]

We kunnen dus zeggen dat MDA een aanpak is voor model gedreven softwareontwikkeling, welke met behulp van modellen het te maken systeem met al zijn componenten en hun relaties beschrijft en genereert. Bij OMG zeggen ze dat MDA een aanpak biedt dat:

- een systeem specificeert onafhankelijk van het platform;
- de platformen specificeert;
- verschillende platformen gekozen kan worden voor het systeem;
- de systeem specificatie getransformeerd kan worden naar een platform specificatie. [MDA]

Als we kijken naar de verschillende modellen die voor MDA voorgeschreven zijn zien we dit ook terug. Het Platform Independent Model (PIM) specificeert een systeem onafhankelijk van het platform, daarnaast hebben we het Platform Specific Model (PSM) welke wel het platform specificeert. En uiteindelijk worden de modellen getransformeerd naar een platform specificatie, in de meeste gevallen de programmacode. Met platform bedoelen we de programmeertaal, middleware, uitwisselingsformaat, etc., waarmee de applicatie geïmplementeerd gaat worden. Enkele voorbeelden hiervan zijn: CORBA, J2EE, .NET en XML. De drie doelen die OMG wil bereiken met MDA zijn portabiliteit, interoperabiliteit en herbruikbaarheid door gebruik te maken van scheidingen van architectuurconcerns. [MDA]

OMG is een organisatie die allerlei standaarden vastlegt en daarbij de doelstelling heeft een richtlijn aan te geven in plaats van een methode die van begin tot eind gevolgd moet worden. Dit houdt in dat een ieder die MDA wil gebruiken en het niet 100% wil navolgen dit kan doen. Om de aanpak nog MDA te noemen zal er gekeken worden naar de specifieke eigenschappen van MDA. Zo moet er gekeken worden of de aanpak een model gedreven softwareontwikkeling is, waarbij de verschillende modellen (PIM, PSM en code) qua abstractie en specificatie van elkaar verschillen. Met andere woorden het PIM mag geen architectuurbeslissingen of platformspecificaties bevatten, wat wel terug te vinden is in het PSM.

[4] http://www.omg.org

2.2 Model Driven (Software) Development

Als er gesproken wordt over MDA dan wordt dit wel eens verward met de methode Model Driven (Software) Development (MDD of MDSD) [MDSD]. MDA is meer een paradigma dan een methodologie en waarbij men zich houdt aan de standaarden in plaats van het beschrijven van een nieuwe ontwikkelingsaanpak. Hierbij kan de ontwikkelaar voor MDA tools zijn eigen aanpak definiëren en door gebruik te maken van de voorgeschreven MDA standaarden het een MDA tool noemen. [MDA]
Beide software ontwikkelingsmethodes zijn model gedreven en hebben veel overeenkomsten. Zo ontwikkelen ze beide met behulp van modellen, welke het toekomstige systeem beschrijven. Ook wordt er vanuit modellen code gegenereerd, om zodoende deze te compileren naar de platform specificatie. Alleen bij MDD wordt er geen platform onafhankelijk model gemaakt, maar begint men meteen met het modelleren van de PSM modellen. [Völter 04]

MDD bestaat al een geruime tijd, enkele voorbeelden uit de jaren ’80 zijn CASE-tools. Deze konden raamwerken voor platformen en code voor het maken van een database genereren. Het genereren van raamwerken is gefaald omdat bedrijven daar geen toekomst in zagen. Maar de CASE-tools voor het genereren van databasecode worden hedendaags nog volop gebruikt. [Greenfield 03]

2.3 Unified Modeling Language

De modellen die gebruikt worden voor MDA zijn, in principe, gemaakt met behulp van UML. UML staat voor Unified Modeling Language. Dit is een algemene visuele modelleringstaal, welke wordt gebruikt om te specificeren, visualiseren, construeren en documenteren van systemen die ontwikkeld moeten worden. In UML wordt zowel de statische structuren als de dynamische gedrag van het systeem opgeslagen. De statische structuur beschrijft de objecten waaruit het systeem bestaat en welke relatie ze onderling met elkaar hebben. Het dynamische gedrag beschrijft de geschiedenis van de objecten over een tijdsbestek en de communicatie tussen objecten om doelen te bereiken [Rumbaugh 99]. UML heeft verschillende modellen om de statische en dynamische aspecten te tonen.

Statische Diagrammen
Class Diagram
Object Diagram
Component Diagram
Deployment Diagram

Dynamische Diagrammen
Use Case Diagram
Sequence Diagram
Collaboration Diagram
Statechart Diagram
Activity Diagram

Tabel 1: Overzicht UML modellen [UML]

Elk model geeft weer een ander beeld van de applicatie, bijvoorbeeld het Class Diagram laat alle klassen zien met hun attributen en methoden. Het Sequence Diagram geeft daarentegen de interactie van gebeurtenissen tegenover de tijdas aan.

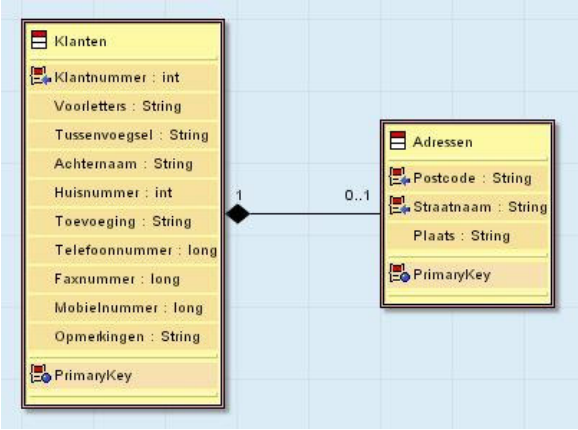
MDA maakt dus gebruik van UML, maar er is een verschil tussen een MDA model en een UML model. Namelijk het feit dat een MDA model gemaakt moet worden met een mate van precisie en volledigheid om uiteindelijk naar code getransformeerd te worden. Een UML model hoeft daar niet aan te voldoen en wordt ook niet getransformeerd naar een model of code.

2.4 Computation Independent Model

MDA gebruikt verschillende modelniveaus om tot de uiteindelijke applicatie te komen. Elk niveau beschrijft het te maken systeem op een bepaalde abstractie. Op het hoogste abstractieniveau zien we het Computation Independent Model (CIM), ook wel het business onafhankelijk model genoemd. Deze beschrijft alle requirements van het systeem, in welke omgeving het systeem gaat draaien en wat er van het systeem wordt verwacht. [MDA] Met de onafhankelijkheid die hier genoemd wordt bedoelt men dat het model onafhankelijk van de implementatie is en niet de platformonafhankelijkheid zoals sommige programmeertalen ondersteunen. Vaak wordt het CIM in MDA tools niet meegenomen omdat men het systeem meteen in het Platform Onafhankelijk Model wil modelleren.

2.5 Platform Independent Model

Op een abstractieniveau lager komen we het Platform Independent Model (PIM) tegen. Het platform onafhankelijk model wordt gebruikt om alle business requirements weer te geven. Het gehele systeem is in een abstracte manier af te lezen. Dit houdt in dat men geen architectuur beslissingen en platformspecificatie kan aflezen uit het model. Zodoende kan men het model gebruiken voor meerdere soorten architecturen en implementatieplatformen/-talen. Het onderhouden van het PIM is makkelijker zonder dat men hoeft te letten op de architectuur en/of implementatie. In figuur 1 ziet u een voorbeeld van een PIM[5] uit OptimalJ, wat een applicatiestructuur beschrijft voor het registreren van klanten. De twee entiteiten die u ziet, Klanten en Adressen, zijn klassen met hun attributen, bedrijfsregels en operaties. Elk attribuut heeft een abstract dataformaat, wat in het PSM een specifiek dataformaat zal worden. Een "String" in een PIM kan bijvoorbeeld een "VARCHAR(40)" in een PSM worden. U ziet het woord PrimaryKey staan, dit is een naam voor een voorgedefinieerde bedrijfsregel. Hiermee wordt in het model vastgelegd dat een bepaald veld, in dit geval het veld "Klantnummer", ten alle tijden op uniekheid moet worden gecontroleerd. Dit ongeacht of er aan de uiteindelijke applicatie een relationele database hangt of niet. Bij de transformatie kan deze informatie ook gebruikt worden voor de identificerende velden in de database. Zodoende wordt er in de applicatie gecontroleerd of er geen redundantie in de tabellen ontstaan, buiten de controle van de database-engine. De relatie tussen de klassen is net zoals in het UML gedefinieerd wordt. Hieraan kan men niet aflezen of dit voor een J2EE platform is, de applicatie een 3-lagen architectuur heeft en/of wat voor soort database-engine er zal draaien.



Figuur 1: PIM in OptimalJ, Registratie van klanten

2.6 Platform Specific Model

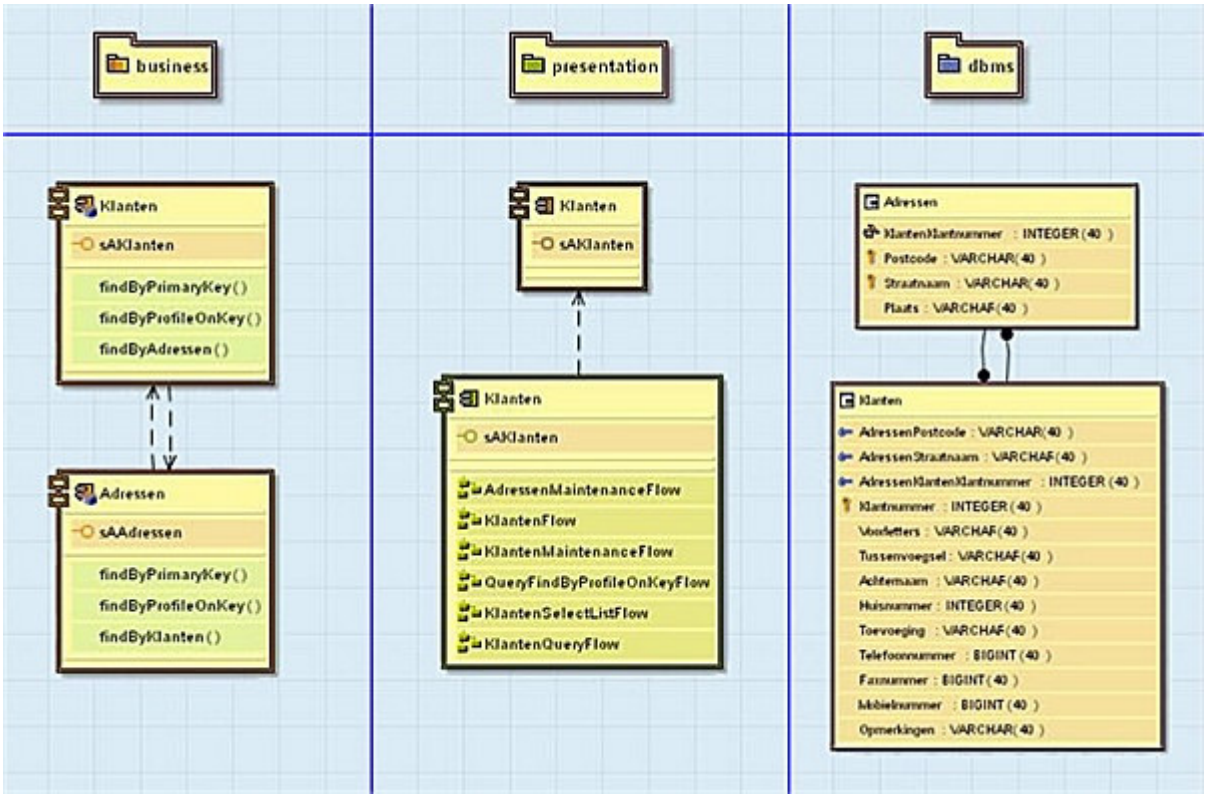
Om van het platform onafhankelijk model naar de uiteindelijke code te komen zal men eerst de architectuurspecificatie in het model moeten verwerken. Om te voorkomen dat het PIM overschreven wordt transformeert men het PIM naar een Platform Specific Model (PSM).

Het PSM bestaat uit alle functionaliteiten van het PIM met alle architectuur beslissingen erin verwerkt. Vaak bestaat het PSM uit meerdere soorten modellen, welke een specifieke view op het systeem weergeven. Zo is er een model die de databasestructuur beschrijft of een model welke de presentatielaag, de stappen die een gebruiker moet doorlopen, beschrijft.
Zo heeft een PSM eigenschappen die een PIM niet heeft, zoals:

- Technologie: taalspecifieke eigenschappen (bijvoorbeeld: Object georiënteerd programmeertalen) of database toegangsmechanismen.
- Communicatie: de beschrijving hoe een component gegevens uitwisselt met andere componenten.
- Interactie: de berichten tussen verschillende componenten en de manier hoe ze elkaar aansturen of van informatie voorzien.
- Foutafhandeling
- Design patterns
- Interface beschrijving
- Beveiliging

Over het algemeen zullen niet alle punten van een PIM naar één PSM getransformeerd worden, maar terug te vinden zijn in meerdere PSM's. Vaak past de ontwikkelaar het PSM op diverse punten aan om het applicatie zo optimaal mogelijk te laten genereren.

[5] Let op, niet elke PIM hoeft als het voorbeeld eruit te zien. Dit is afhankelijk van de toolontwikkelaar.



Figuur 2: Overzicht modellen van het PSM van MDA tool OptimalJ

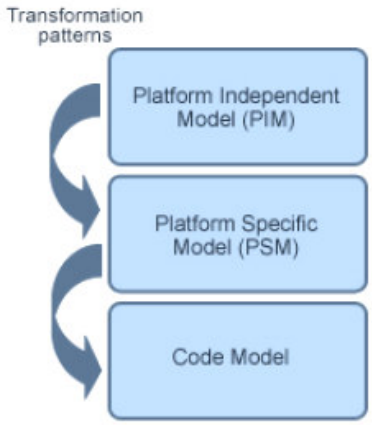
2.7 Code Model

Om uiteindelijk tot een gegenereerde applicatie of code te komen zal men vanuit PSM de code moeten genereren. De code is dan het allerlaatste abstractieniveau voor de ontwikkelaar, hierna zal de code gecompileerd worden waarna de applicatie klaar is voor testen en hopelijk het gebruik. Uiteraard heeft de ontwikkelaar nog de mogelijkheid delen van de gegenereerde code te herschrijven indien het niet volledig of niet naar wens is. Een belangrijk aspect voor een MDA tool moet dan ook zijn dat de code die door een programmeur geschreven is bij het opnieuw transformeren van de modellen niet verloren gaat.

2.8 Transformaties

Bij MDA staan de (automatische) transformaties tussen de modellen en code centraal. 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. [Kleppe 03]

In de MDA standaard wordt een transformatiedefinitie een transformatie pattern genoemd. Welke dus beschrijft hoe bijvoorbeeld het PIM naar het PSM getransformeerd moet worden.



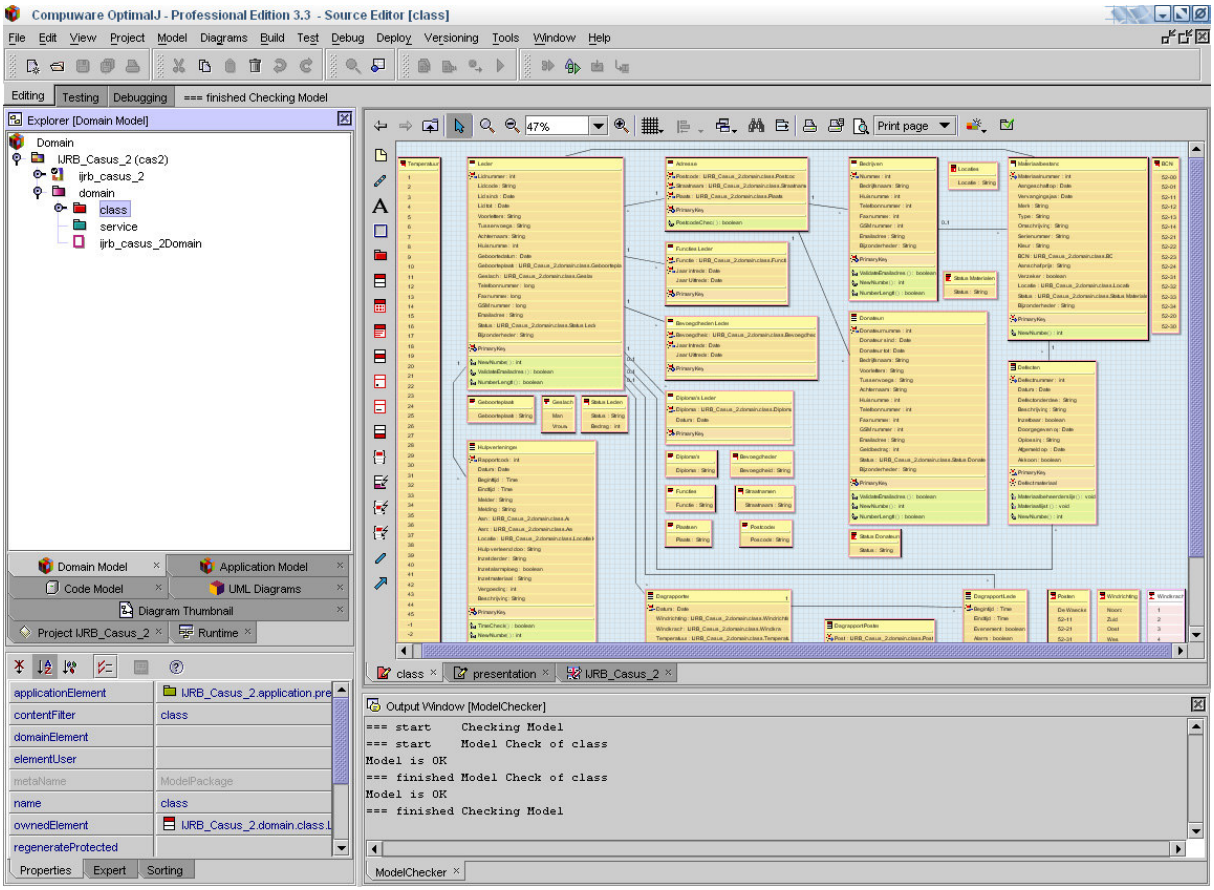
Figuur 3: MDA transformaties [OptimalJ]

3 OptimalJ 3.3 Professional Edition

Het MDA tool welke onderzocht wordt is genaamd OptimalJ 3.3 Professional Edition. Dit software ontwikkelingsprogramma is ontwikkeld door het bedrijf Compuware[6]. Compuware is van oorsprong een Amerikaans bedrijf dat, door overname van het Nederlandse bedrijf UNIFACE, in 1994 een vestiging in Nederland kreeg. In het jaar 1999 zijn bij Compuware allerlei interne discussies opgestart om een J2EE gebaseerde ontwikkelingomgeving te hebben. Dit naar aanleiding van de populariteit van Java. De nieuwe ontwikkelomgeving werd OptimalJ genoemd en MDA is vanaf het begin het uitgangspunt geweest als kern architectuur van deze ontwikkelomgeving. Het eerste 'controlled release' van OptimalJ is uitgebracht in augustus 2001. Op 14 juli 2005 heeft men versie 4.0 uitgebracht.

3.1 Omgeving en Lay-out

Figuur 4 geeft u een impressie hoe OptimalJ eruit ziet als applicatie. De gehele applicatie is met Java Technologie gebouwd en ondersteunt het Windows uiterlijk. De applicatie is voorzien van menustructuren, werkbalken, boomstructuren voor een overzicht van de modellen, een informatie-gedeelte, diversen functionaliteiten en een algemeen modelleergebied waarbij de modellen van elkaar worden gescheiden door middel van tabbladen.



Figuur 4: Overzicht lay-out OptimalJ 3.3 Professional Edition

In de menustructuur zijn alle mogelijkheden van het programma opgenomen, waarbij de belangrijkste functies nog eens extra in de werkbalk terug te vinden zijn. Er zijn drie hoofd-scheidingen binnen OptimalJ, namelijk het bewerkingsgedeelte, het testgedeelte en het fout-opsporingsgedeelte. Deze zijn terug te vinden net onder de werkbalk en zien eruit als een tabblad-selector.

[6] http://www.compuware.com

Gezien het onderzoek betrekking heeft op het bewerkingsgedeelte, zal er niet verder ingegaan worden op hoe de andere gedeeltes eruit zien qua lay-out en functionaliteiten. Compuware heeft zijn best gedaan om elke onderdeel van OptimalJ een eigen identificatie te geven, zodat de gebruiker makkelijker een onderdeel terug kan vinden en/of kan herkennen. Zo heeft niet alleen elk model een uniek icoon, zelfs de verschillende eigenschappen hebben er een. De boomstructuur laat de gebruiker een overzicht zien waaruit het model bestaat, zonder deze op het werkblad te openen. De verschillende modellen zijn door middel van een tabbladselector onderaan het overzichtsvenster, met de boomstructuren, te selecteren.
Het modelleergedeelte heeft zijn eigen werkbalk, welke per soort model qua onderdelen aangepast wordt. Zodoende kan men niet de verschillende modeleigenschappen door elkaar halen bij het modelleren. Voor informatie zoals: foutenmeldingen, informatieberichten en waarschuwingen heeft OptimalJ een apart venstergedeelte wat pas zichtbaar wordt wanneer dat nodig is. Dit kan door de gebruiker weg geklikt worden, waardoor het informatiegedeelte niet meer zichtbaar is tot dat er weer nieuwe informatie komt. Het makkelijke aan de gehele OptimalJ omgeving is dat men in staat is om deze geheel naar eigen wensen en eisen aan te passen. Indien men de diverse gedeeltes op een andere plek wilt hebben is dit mogelijk.

3.2 Modellen

OptimalJ heeft diverse modellen ten behoeve van het genereren van software. Elk model heeft zo zijn eigen functie en doel. Er kunnen vier modellen worden onderscheiden: Domain, Application, Code en UML model. Drie van deze modellen bestaan uit submodellen welke samen het gehele model voorstellen. In OptimalJ heeft het UML model als enige geen directe betrekking tot het MDA proces. Wel kan deze geïmporteerd worden als deel van het MDA model.

Hoofdmodellen	Submodellen	Beschrijving
Domain Model	Class Model	Beschrijft de structuur van de applicatie. Elementen van dit model zijn domeinclasses, hun associaties, verschillende datatypes, attributen en unieke constraints.
	Service Model	Beschrijft informatie in de vorm van domein views en services. De views tonen delen van het class model en de services dienen een view, maar dit is niet verplicht.
Application Model	Presentatie Model	Beschrijft de Web front-end van de applicatie, zowel uiterlijk als de navigatie er doorheen.
	Business Façade Model	Beschrijft de communicatie tussen componenten die zich bevinden in de presentatie- en logicalaag.
	Business Logic Model	Beschrijft de businesscomponenten die betrekking hebben tot datamanipulatie en applicatiegedrag.
	Business Common Model	Beschrijft de gedeelde datastructuren, datatypes, expressies, uitzonderingen, etc..
	DBMS Model	Beschrijft de onderliggende database-structuur.
Code Model	Alle gegenereerde code	Laat alle gegeneerde code zien en geeft de mogelijkheid om deze in de vrijgegeven gedeeltes aan te passen. Dit model beschermt tevens de eigen geschreven code.
UML Model	Class Diagram	Diversen statische en dynamische UML modellen welke los staan van het MDA proces van OptimalJ. Wel kan men de UML modellen gebruiken als Domain of Application Model, afhankelijk van in hoeverre OptimalJ met de gegevens kan werken.
	Object Diagram	
	Component Diagram	
	Deployment Diagram	
	Use Case Diagram	
	Sequence Diagram	
	Collaboration Diagram	
	Statechart Diagram	
	Activity Diagram	

Tabel 2: Informatie over modellen in OptimalJ, op basis van [OptimalJ]

De domein-, applicatie- en codemodellen hebben onderling als doel aanvullend voor elkaar te zijn en worden aan elkaar verbonden door middel van patterns. De gedachtengang is wel dat men een model aanpast om de uiteindelijke applicatie daarmee ook aan te passen.
Elk model heeft zijn eigen abstractieniveau en de gebruiker zal, aan de hand van wat aangepast moet worden, moeten bepalen in welk model dat zal gebeuren. Als men het Domain Model wijzigt dan worden deze wijzigingen doorgevoerd naar alle applicatie modellen, is er dus een wijziging voor het gehele systeem zal dit mogelijk gebeuren in het Domain Model. In het Application Model daarentegen wijzigt men alleen de specifieke eigenschappen van de applicatie ten behoeve van het platform, dit kan betrekking hebben op één van de applicatie modellen. Wijzigingen in het Code Model hebben betrekking op de implementatie van complexe procedurele logica, zoals de body van een BusinessMethod. [OptimalJ]

3.3 Functionaliteiten

OptimalJ bied veel functionaliteiten zoals, het beschermen van eigen geprogrammeerde code, integratie met andere IDE's, de ondersteuning voor versiebeheer, testen en debuggen van de applicatie en het daarvoor controleren van de modellen en code die gemaakt en/of gegenereerd zijn.

Zoals al eerder genoemd kan men in het code model op bepaalde plaatsen nieuwe code erbij programmeren. Deze plaatsen worden Free Blocks genoemd en worden in de tekstlay-out gemarkeerd door middel van blauwe of witte blokken. De blauwe blokken zijn beschermde stukken code, welke door OptimalJ gegenereerd zijn en niet aangepast kunnen worden. De witte blokken markeren de stukken code (of open ruimtes in de code) waar de programmeur functies kan uitprogrammeren of de applicatie kan bijsturen op codeniveau. Indien de applicatie opnieuw gegenereerd dient te worden vormt dit geen probleem voor de geprogrammeerde code. Zoals de gegenereerde code beschermd wordt tegen aanpassingen van de programmeur wordt de eigen geschreven code beschermd tegen wijzigingen van het automatische proces. Kortom na updaten en hergenereren van de applicatie staat de eigen geschreven code nog steeds ongewijzigd op dezelfde plaats.

```
public boolean postcodeCheck(java.lang.String postcode)
    throws com.compuware.alturadev.application.AlturaPreConditionException, com.compuware.alturadev



    boolean  returnValue = false;

    // if this method is making any changes to bean managed persisted data (e.g. aggregated data),
    // this can be signaled to the application server by uncommenting the following line:
    // setIsDirty(true);



    return returnValue;

}
```

Figuur 5: Een deel gegenereerde code met beschermde en onbeschermde blokken

Andere functionaliteit is dat de programmeurs in hun eigen Integrated Development Environment (IDE) kunnen blijven werken zonder de mogelijkheid om de beschermde code te kunnen aanpassen. OptimalJ ondersteunt namelijk de mogelijkheid dat de (OptimalJ)architect de applicatie ontwikkelt en beschikbaar maakt voor het ontwikkelteam. Zij kunnen dan in hun vertrouwde omgeving blijven werken, terwijl de applicatie verder geprogrammeerd wordt met de flexibiliteit van de MDA architectuur. OptimalJ is geïntegreerd met de volgende IDE's:

- Eclipse
- IntelliJ IDEA
- JBuilder (versie 9 en X)
- WebSphere Application Developer (WSAD)
- WebLogic Workshop

OptimalJ ondersteunt niet alleen de integratie met IDE's maar ook de migratie met een versie-beheersysteem. Zodoende kan het ontwikkelteam dat in OptimalJ werkt altijd weer terug gaan

naar een vorige versie en/of gebruik maken van alle andere functionaliteiten van versiebeheer-systeem. Naast het beschermen van de code, de integratie van IDE's en migratie van versie-beheersysteem doet OptimalJ ook regelmatige controles van de modellen en code. Voordat het Application Model gegenereerd wordt, is er een controle van het Domain Model. Dit wordt bij de generatie van het code model weer gedaan.

De ontwikkelaar is tevens ook in staat de applicatie vanuit OptimalJ te testen middels een applicatie- en databaseserver. Om de databaseserver te starten moet men de database SOLID handmatig starten, waarna men via het code model de betreffende scripts kan inladen. Via OptimalJ is het wel mogelijk om de databaseconsole te openen. Nadat men de database in orde heeft gemaakt kan er overgegaan worden tot het starten van de applicatie. Met behulp van de applicatieserver, welke de gecompileerde code zal uitvoeren en het resultaat zal tonen.

3.4 Installatie-eisen

Voordat OptimalJ geïnstalleerd kan worden, moeten de systeemeisen goed bekeken worden. Een minimale systeemeis is bijvoorbeeld de installatie van de Java SDK 1.4 omgeving. Is deze niet geïnstalleerd dan zal de installatie van OptimalJ niet doorgaan. Dit is omdat zowel de installer als OptimalJ in Java werken.

OptimalJ heeft de volgende minimale systeemeisen voor 32-bits Windows systemen:
- Microsoft Windows 2000 or XP
- 512 MB of physical memory
- CPU speed 1 GHz
- Screen resolution: 1024 x 768
- 800 MB of free disk space

Voor de Red Hat Linux Enterprise Edition 3.0 systeem gelden de volgende systeemeisen:
- Red Hat Linux Enterprise Edition 3.0
- JAVA_HOME environment variable set to the java install dir
- Graphical Desktop Manager (KDE)
- 512 MB of physical memory
- CPU speed 1 GHz
- Screen resolution: 1024 x 768
- 800 MB of free disk space

Ook kan men OptimalJ gebruiken als plug-in van andere IDE's, zoals:
- JBuilder plug-in vereist de installatie van JBuilder 10
- IBM WSAD plug-in vereist de installatie van WebSphere Studio Application Developer (Windows) Version 5
- Eclipse plug-in vereist de installatie van Eclipse Platform 2.1.3 or 3.0.1.

Het is een pluspunt om te zien dat OptimalJ zowel te installeren is op Windows als Linux en te gebruiken is in andere IDE omgevingen. Voor dit onderzoek is OptimalJ alleen op Windows geïnstalleerd.

3.5 MDA in OptimalJ

Compuware claimt dat OptimalJ een MDA tool is. Om dit te kunnen vaststellen moet er eerst gekeken worden of er MDA kenmerken, zoals beschreven in hoofdstuk 0, in OptimalJ te vinden zijn. Op het eerste gezicht lijken de modellen, zoals MDA ze omschrijft, niet aanwezig te zijn. Dit blijkt niet te kloppen omdat Compuware andere benamingen voor de modellen heeft gebruikt. In tabel 3 kunt u aflezen welke modellen van OptimalJ overeenkomen met die van MDA.

OptimalJ 3.3 PE	Model Driven Architecture
-	Business Onafhankelijk Model (CIM)
Domain Model	Platform Onafhankelijk Model (PIM)
Application Model	Platform Afhankelijk Model (PSM)
Code Model	Code Model

Tabel 3: OptimalJ modellen versus MDA modellen [OptimalJ]

Om vast te stellen dat de modellen werkelijk overeenkomen met de definitie van de MDA modellen moet er gekeken worden naar de overeenkomsten op een gedetailleerder niveau. Het CIM is in

MDA geen verplicht model en wordt ook niet altijd gebruikt. Deze is dan ook niet terug te vinden in OptimalJ. Voor het PIM geldt dat het model platform onafhankelijk moet zijn. Als we gaan kijken naar het Domain Model dan blijkt dit te kloppen, er zijn geen duidelijke platform specifieke kenmerken in het model te vinden. Er is geen architectuur te vinden en het beschrijft de applicatie zonder dat daaruit valt op te maken dat het voor C++, Java of een bepaalt soort database geschreven is.
De volgende stap is het vaststellen of PSM en het Application Model met elkaar gelijk zijn. Hierbij wordt niet gekeken naar de transformatie van het PIM naar het PSM, maar alleen naar het platform specifiek model. In het PSM zouden we informatie van het PIM met architectuurbeslissingen terug moeten vinden. Het Application Model heeft daadwerkelijk alle informatie van het Domain Model. Deze informatie is verwerkt in een gekozen architectuur, in dit geval een drie lagen architectuur. En er zijn ook nog platform specifieke eigenschappen in het model te vinden.
De code is dan ook het laatste abstractieniveau. Op dit niveau kan men logica (her)programmeren en uiteindelijk door middel van een compiler naar de werkelijke applicatie omzetten. Het code model in OptimalJ kan worden bekeken zowel tekstueel niveau als modelniveau. Op het model-niveau kan men de code in de vorm van een klassenmodel bekijken. Tevens kan de ontwikkelaar ook de code, op tekstueel niveau, bijwerken.

Nu is er nog niet gekeken naar de transformaties van het ene model naar het andere. Dit is ook een belangrijk aspect van het MDA proces. Om de transformatie uit te kunnen voeren heeft men transformatiedefinities nodig, welke de regels en specificaties beschrijft om van het bronmodel naar het doelmodel te komen. In OptimalJ gebeurt dit door middel van patterns die de regels en specificaties bevatten. Vooralsnog kan men alleen op het J2EE platform transformaties uitvoeren. Hier vanuit kan de platform onafhankelijkheid van het OptimalJ proces ter discussie worden gesteld. Voor het onderzoek is gebruik gemaakt van de Professional Edition van OptimalJ, echter aangezien men hierin niet zelf patterns kan definiëren levert dit geen bewijs voor de bovenstaande stelling. In een gesprek met Compuware, ten behoeve van dit onderzoek, is de vraag gesteld of het mogelijk is het domain model naar bijvoorbeeld C++ application model te transformeren. Men antwoordde dat dit standaard niet kan, maar door het maken van een speciale pattern het wel mogelijk gemaakt kan worden. Hiervoor heeft de ontwikkelaar wel de Architecture Edition van OptimalJ nodig. Er kan gesteld worden dat, theoretisch, er voor verschillende platformen een application model gegenereerd kan worden met OptimalJ, mits er een aparte pattern daarvoor wordt geschreven. Voor het genereren van databasecode zijn er wel meerdere mogelijkheden binnen OptimalJ. De ontwikkelaar kan kiezen uit verschillende databasesoorten waar de toekomstige applicatie zijn gegevens uit kan halen of kan verwerken.
Op het gebied van architectuur kan de gebruiker uit verschillende soorten architecturen kiezen. Deze architecturen verschillen voornamelijk in het aantal lagen en de oriëntatie qua classes, webservices of integratiekoppelingen.

Over het geheel gezien kunnen we concluderen dat OptimalJ een MDA tool is. Het gebied van meerdere platformen kan worden verbeterd. Compuware heeft laten weten dat er in de toekomst een OptimalJ versie uitkomt die ook transformaties voor .NET aankan.

3.6 Schaalbaarheid van modellen in OptimalJ

Het onderzoek betreft voornamelijk de schaalbaarheid van de modellen in OptimalJ met betrekking tot het geheugen. Eerst is er op theoretisch vlak gekeken of Compuware aangeeft dat er problemen of grenzen op dit gebied liggen. Dit blijkt in de handleiding van OptimalJ te worden aangegeven. Er staat dat de grootte van de modellen samenhangt met de grootte van het geheugen wat in het systeem zit. [OptimalJ]
Indien de computer crasht, door een capaciteitsprobleem van het geheugen, dient men eerst de .ini bestanden van OptimalJ te wijzigen om een volgende crash van de computer te voorkomen. Als dit niet werkt zal de ontwikkelaar over moeten gaan tot het vergroten van zijn werkgeheugen. Wat ontbreekt is een indicatie hoe groot het model kan zijn tegenover de hoeveelheid geheugen voordat de computer het model niet meer aankan.

Vanwege het ontbreken van deze gegevens wordt het onderzoek daarop toegespitst. Er gaat onderzocht worden hoe groot het model kan zijn en/of er verbanden zijn met de grootte van het model en het geheugen van de computer.

4 Resultaten Casussen

De casussen zijn gebaseerd op een al bestaand stuk software, dat geschreven is voor de IJmuider Reddingsbrigade[7] (IJRB). De software bestaat uit verschillende administratieve taken, van het administreren van leden tot de hulpverleningen die de IJRB op het strand doet. Gezien de diversiteit en de omvang van software kunnen we spreken over een middelgroot softwarepakket.

4.1 Casus 1

Deze casus beschrijft één gedeelte van het gehele softwarepakket van de IJRB. Dit gedeelte is namelijk de administratie van de leden met hun functies, bevoegdheden en diploma's. Voor meer informatie betreffende de casus en een visuele afbeelding wordt u doorverwezen naar bijlage A, Casus 1.

Bij het modelleren van de casus in OptimalJ werd er tegen een stuk onbekendheid opgelopen. Dit komt omdat het modelleren van MDA verschilt van het modelleren in UML, bijvoorbeeld door het gebruik maken van een klassendiagram. Door het lezen van de handleiding, het uitvoeren van tutorials en hulp van medewerkers van Compuware kon dit probleem opgelost worden.
In tabel 4 kunnen we lezen wat het geheugenverbruik tijdens het modelleren, transformeren en andere OptimalJ activiteiten is geweest. Dit geeft ons een goed beeld wat er gebeurd met het geheugen van de computer tijdens het toepassen van een MDA-tool.

Actie		Tijdsduur (mm:ss:ms)	Wisselbestand (KB)	Fysiek geheugen (KB)	%
Modelleren					
	Entiteit 1	-	573440	396888	80
	Entiteit 2	-	578560	411472	83
	Entiteit 3	-	572416	412544	84
	Entiteit 4	-	573440	423004	86
Importeren model Casus 1		-	592792	453220	92
Transformatie naar PSM		00:44:16	603540	411572	83
Na transformatie PSM		-	601628	451880	92
Genereren van code		01:11:77	629292	465540	94
Na genereren code		-	602484	468028	95
Building van applicatie		01:26:06	606884	446060	90
Na building applicatie		-	602940	437276	89

Tabel 4: Geheugenverbruik en tijdregistratie tijdens maken van casus 1

In de tabel ziet u vijf kolommen, waarin diverse gegevens vermeld staan. Gegevens zoals actie, tijdsduur en geheugenverbruik. Indien het fysieke geheugen 100% zou zijn, dan is er een mogelijkheid dat de computer crasht. Dit sluit niet uit dat de computer in andere situaties niet kan crashen. In de kolom "Actie" ziet u bij modelleren een onderverdeling van entiteiten. Dit is gedaan om te zien hoe het geheugen zich gedraagt tijdens het modelleren.

Aan het einde van de uitvoering van deze casus kwam het gewenste resultaat, namelijk het administreren van leden met aanverwante informatie. Wat opviel is dat het uitrekenen van een leeftijd of hoe lang iemand lid is niet standaard gegenereerd kan worden. Alleen het raamwerk met de in- en uitgaande variabelen wordt gegenereerd, de logica dient de ontwikkelaar te programmeren. Bij het testen van de applicatie bleek dat er verschillende delen van de applicatie niet correct werkte. Zo kon er geen lid ingevoerd worden omdat het bijbehorende adres verplicht is. Het invoeren van een adres ging niet omdat het niet opgeslagen werd. De oorzaak hiervan is onbekend, mogelijk in de applicatieserver, en valt buiten de scope van dit onderzoek.

[7] http://www.ijrb.nl

4.2 Casus 2

Deze casus is een uitbreiding van casus 1. Hiermee wordt het model opgeschaald naar een groter geheel. De uitbreiding betreft de volgende administratieve taken, welke zijn onderverdeeld om het onderzoek beter te structureren:

- Deel 2a: Administreren van donateurs, sponsoren, bedrijven en leveranciers;
- Deel 2b: Bijhouden van het materiaal dat de IJRB bezit en de defecten eraan;
- Deel 2c: Urenregistratie van de strandploeg en de hulpverleningen per dag;

Voor meer informatie betreffende de casus en een visuele afbeelding wordt u doorverwezen naar bijlage A, Casus 2.

Gezien de ervaring die opgedaan is in casus 1 is onbekendheid in het modelleren van MDA modellen in deze casus geen probleem. Extra tijd kostte het goed overdenken welk soort datatype waar te gebruiken. Een aantal keer was de computer traag, maar is niet vastgelopen.
Het geheugenverbruik en de tijdsduur zijn in tabel 5 af te lezen. Deze tabel is alleen iets anders af te lezen dan die van casus 1. Dit wordt veroorzaakt door de delen van de casus, welke een duidelijk verloop laten zien van de verschillende groottes. Helaas is daardoor niet goed af te lezen wat het verloop is van de eerste actie tot de laatste.

Actie		Tijdsduur (mm:ss:ms)	Wisselbestand (KB)	Fysiek geheugen (KB)	%
Modelleren					
	Deel 2a	-	598464	451812	92
	Deel 2b	-	599232	453124	92
	Deel 2c	-	601648	456664	93
Importeren model Casus 2					
	Deel 2a	-	595044	443324	90
	Deel 2b	-	603788	441540	90
	Deel 2c	-	606128	450048	91
Transformatie naar PSM					
	Deel 2a	00:57:38	603436	448564	90
	Deel 2b	01:15:26	606484	478176	97
	Deel 2c	01:45:69	607680	468880	95
Na transformatie PSM					
	Deel 2a	-	602800	456880	93
	Deel 2b	-	609688	471880	96
	Deel 2c	-	604860	479936	97
Genereren van code					
	Deel 2a	01:22:66	627444	463344	94
	Deel 2b	01:44:70	634444	461716	94
	Deel 2c	02:23:46	615312	462680	94
Na genereren code					
	Deel 2a	-	596552	472604	96
	Deel 2b	-	631112	468328	95
	Deel 2c	-	606516	467900	95
Building van applicatie					
	Deel 2a	01:47:16	602808	446448	91
	Deel 2b	02:41:96	611704	448216	91
	Deel 2c	03:50:16	629744	460312	93
Na building applicatie					
	Deel 2a	-	597368	437944	89
	Deel 2b	-	605600	441240	90
	Deel 2c	-	607136	460244	93

Tabel 5: Geheugenverbruik en tijdregistratie tijdens maken van casus 2

Aan het einde van deze casus was het gewenste resultaat zichtbaar. Ook hierbij de opmerking dat bij meer tijd te besteden aan beter modelleren, het resultaat ook beter wordt. Voor het onderzoek is het niet relevant dat het resultaat 100% overeenkomt met de werkelijke software. Net als bij casus 1 werkte de applicatie bij het testen niet volledig.

4.3 Casus 3

Deze casus is een uitbreiding van de vorige casussen. Hierbij ging het niet direct om de schaalbaarheid van het model, maar meer om de logica in de applicatie. De uitbreiding betreft het uitrekenen van diverse soorten statistieken, zoals bijvoorbeeld het tellen van het aantal keer dat een strandploeglid ingeschreven staat of hoeveel leden lid zijn. Ook voor deze casus kunt u meer informatie vinden in bijlage A.

Tijdens het uitvoeren van de andere casussen werd al duidelijk dat er geen simpele logica, zoals het optellen van twee velden, gegenereerd kan worden. OptimalJ is in deze gevallen alleen in staat raamwerken te genereren. Desondanks is casus 3 toch uitgevoerd om de schaalbaarheid van de modellen in kaart te brengen. Hiervoor moest er een klasse gemodelleerd worden met verschillende operaties, maar zonder attributen. De verzamelde gegevens kunt u terug lezen in tabel 6. Gezien deze casus geen delen bevat is het verloop van modelleren tot het compileren van de applicatie af te lezen.

Actie	Tijdsduur (mm:ss:ms)	Wisselbestand (KB)	Fysiek geheugen (KB)	%
Modelleren	-	608772	465267	94
Importeren model	-	608512	449556	91
Transformatie naar PSM	01:46:32	620207	453972	92
Na transformatie PSM	-	619120	431968	88
Genereren van code	02:26:30	624312	462566	94
Na genereren code	-	617496	441254	90
Building van applicatie	03:51:48	640045	462443	94
Na building applicatie	-	603288	459856	93

Tabel 6: Geheugenverbruik en tijdregistratie tijdens maken van casus 3

Net zoals in de ander casussen werd alleen het raamwerk gegenereerd. Dit houdt in dat de ontwikkelaar de logica zelf nog moet programmeren. Gezien het programmeren van de logica niet tot de scope van dit onderzoek behoort is dit niet uitgevoerd. De verschillen met casus 2 zijn aanwezig, hoewel deze niet groot zijn. Alleen na het transformeren van het PIM naar het PSM is een afwijking te zien, dit is onverklaarbaar.

Er is contact geweest met Compuware over het feit waarom zelfs een simpele optelling van twee velden niet gemodelleerd kon worden. Compuware gaf aan dat dit beter kan en dit in de toekomst zeker één van de verbeterpunten zal zijn. Op dit moment konden ze geen oplossing hiervoor bieden. Het schijnt een bekend struikelblok te zijn in MDA waardoor er niet vanuit mocht worden gegaan dat MDA een sluitend geheel zou bieden. In de toekomstige versies van OptimalJ zal er meer aandacht besteedt worden om simpele dataoperaties te kunnen genereren.

5 Bevindingen OptimalJ

In dit hoofdstuk zal er aandacht gegeven worden aan het beantwoorden van de onderzoeksvragen. Naast de vooraf opgestelde onderzoeksvragen, zijn er ook punten die als aandachtspunt naar voren gehaald zullen worden. Deze zijn gebaseerd op waarnemingen en verzamelde gegevens tijdens het onderzoek.

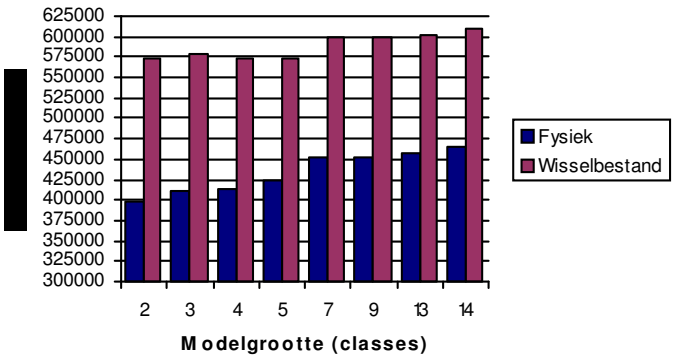
5.1 Installatie

Het installeren van OptimalJ is één van de aandachtspunten. Voor het installeren van OptimalJ moet de omgeving/computer namelijk eerst voldoen aan de minimale systeemeisen. Bij het testen en controleren van dit punt werd geconstateerd dat er volledig aan de minimale eisen moet worden voldaan. Is dit niet het geval dan wordt het programma wel geïnstalleerd maar kan het niet worden gestart. Dit is een opmerkelijk verschil met de meeste andere softwareprogramma's, welke geïnstalleerd en gestart kunnen worden met een systeem dat niet aan de minimumeisen voldoet. Meestal is dan de performance van dat soort programma's slecht. De strenge systeemeisen zijn een pluspunt, omdat de gebruiker door de controle van de minimale eisen gegarandeerd is van goed werkende software.
Een ander punt bij het installeren is het geïnstalleerd hebben van de juiste Java versie. Als je de juiste Java versie hebt voor het installatieproces dan hoeft dit niet de juiste versie te zijn voor het starten van OptimalJ. Tijdens het onderzoek is er tijd verloren om de juiste Java versie te installeren en configureren. Voor de installatie is er namelijk Java versie 5 gebruikt, maar bij het starten van OptimalJ kan alleen maar versie 4.7 worden gebruikt. Dit is ervaren als een minpunt van het softwarepakket, omdat we verwachten dat er ondersteuning wordt gegeven voor versie 4.7 en hoger. Zodoende hoeft de ontwikkelaar alleen te zorgen voor de nieuwste Java versie op zijn systeem, in plaats van verschillende versies.

5.2 Schaalbaarheid van de modellen

De schaalbaarheid van de modellen is één van de belangrijkste onderzoeksgebieden in dit onderzoek. Er wordt hierbij gekeken naar hoe groot het model gemaakt kan worden met de minimale systeemeisen. Hier vanuit kan vastgesteld worden hoe MDA tools met het geheugen omgaan en hoeveel geheugen een ontwikkelaar nodig heeft om een groot softwarepakket te ontwikkelen.

Tijdens het uitvoeren van de casussen zijn er verschillende gegevens verzameld over het geheugenverbruik. Deze gegevens zijn terug te lezen de tabellen van de casussen in hoofdstuk 4. De gegevens over het geheugenverbruik tijdens het modelleren zijn verwerkt in grafiek 1. Zoals in de grafiek af te lezen is, zijn de waardes vanaf modelgrootte 7 en 13 een stuk hoger dan de lijn van verwachting. Dit ligt aan het feit dat een nieuwe casus werd uitgevoerd, hiervoor werd OptimalJ opnieuw opgestart waardoor het geheugen van waarde veranderde. Ook de grootte van het wisselbestand is groter in casus 2 en het geheugenverbruik tussen de grootte van modellen is kleiner. Dit komt doordat OptimalJ mogelijk een view maakt van het model, waardoor de geheugenwaarden veranderden. Ondanks deze veranderingen is er een verband te zien tussen de grootte van het model en het geheugenverbruik.



Grafiek 1: Geheugenverbruik tijdens het modelleren

Naar mate het model groter wordt, neemt ook het geheugenverbruik toe. Als we de trend doortrekken, dan kunnen we voorspellen dat bij het dubbel aantal klassen er geen groter model gemaakt kan worden. Hierbij zal de ontwikkelaar gedwongen worden het fysieke geheugen uit te breiden. Helaas moeten we ook constateren dat het verbruik van het geheugen afhankelijk is van hoe groot een klasse is, het aantal relaties dat de klasse heeft en het aantal datatypes dat ervoor gebruikt wordt. Hierdoor is het moeilijk 100% correct te bepalen bij hoeveel klassen het systeem het model niet meer aan kan. Tevens is er nog één variabele in het spel, namelijk het

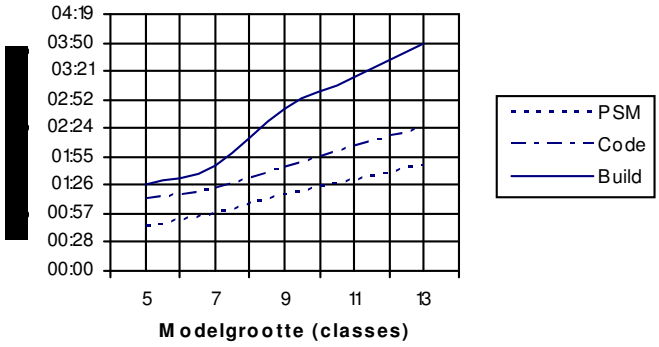
wisselbestand. Als deze kleiner was geweest, dan hadden de geheugengegevens andere waardes gehad en het systeem mogelijk een minder groot model aangekund.

Aan te bevelen is dat een ontwikkelaar 1GB fysiek geheugen in zijn computer heeft wanneer deze gebruik wil maken van OptimalJ. Zodoende is er voldoende geheugen aanwezig om grote modellen te maken. In ieder geval is er bewezen dat met het minimale geheugen een middelgroot model en applicatie door OptimalJ gemaakt kan worden.

5.3 Performance OptimalJ

Tijdens het uitvoeren van het onderzoek en de casussen is er ook gekeken naar de prestaties van de MDA tool. Dit met name op het vlak van de transformatie van het ene model naar het andere model, bijvoorbeeld het PIM naar het PSM.

Bij het uitvoeren van de casussen zijn de tijden bijgehouden over hoelang OptimalJ er, per casus onderdeel, over deed om de transformatie uit te voeren. Dit resulteerde in de gegevens die u kunt aflezen in de tabellen van de casussen in hoofdstuk 4. Vanuit deze gegevens is er een grafiek gemaakt welke een grafische representatie geeft over het verloop van de tijd tegenover de grootte van het model. Dit is af te lezen in grafiek 2. Om de grafiek correct te maken, voor een goede schaalverdeling, is punt 11 berekend. Zodoende krijg men een goed beeld hoe stijl de lijn loopt. Als we een lijn trekken wordt van punt 5 tot 13, bij alle drie de lijnen, dan krijgen we een nominale tijdlijn. Deze lijn geeft een indicatie hoelang OptimalJ over een transformatie zou kunnen doen. Uiteraard is dit niet altijd een vast gegeven. De tijd die voor een transformatie nodig is, wordt mede bepaald door andere invloeden dan alleen de modelgrootte. Deze invloeden zijn bijvoorbeeld: de grootte van de klassen, het aantal gebruikte datatypes en hun relaties met de klassen, de hoeveelheid wijzigingen in het PSM of de hoeveelheid code welke in het code model erbij is geprogrammeerd. In de grafiek is te zien dat de lijn van PSM en code evenredig aan elkaar verlopen. Dit komt doordat in het onderzoek de PSM niet aangepast is. We kunnen zeggen dat indien het PSM bijgewerkt/veranderd wordt deze lijnen niet meer evenredig aan elkaar lopen en het compileren mogelijk ook langer zal duren.



Grafiek 2: Prestaties OptimalJ tijdens de transformaties

Hieruit kunnen we concluderen dat hoe groter en ingewikkelder het model wordt, hoe meer tijd er nodig is om deze te transformeren of te builden.

5.4 Kwaliteit van de gegenereerde code

Niet alleen de grootte van modellen die gemaakt kunnen worden en de prestatie van de MDA tool zijn onderzocht, maar ook is er gekeken naar de kwaliteit van de gegenereerde code. In het onderzoek is er gekeken naar de bescherming van de geprogrammeerde en gegenereerde code, de overzichtelijkheid van de code en hoe de structuur van de code is.

```
// NOTE: this method is required by EJB 2.0
/**
* Compares the specified object with this entry for equality
* @param that the reference object with which to compare
* @return true if the given object is also a map entry and th
*/
public boolean equals(Object that) {
    if (that instanceof LedenKey) {
        LedenKey thatKey = (LedenKey) that;
        return (thatKey.getUniqueId() == null ? getUniqueId()
    }
    return false;
}
```

Figuur 6: Een deel van de gegenereerde code van OptimalJ

Wat opvalt is dat het zoeken van een bepaalde klasse en/of procedure in het code model niet bepaald makkelijk is. Als men dit middels het applicatie model doet is dit een stuk makkelijker. Hier zoekt men het juiste onderdeel op, klikt erop voor het snelmenu en kiest welk deel van de code getoond moet worden. Als we naar de code kijken, zien we een overzichtelijk en gestructureerd geheel. Tevens zien we ook de blauwe en witte blokken, die aangeven welke delen aanpasbaar zijn en wel niet.

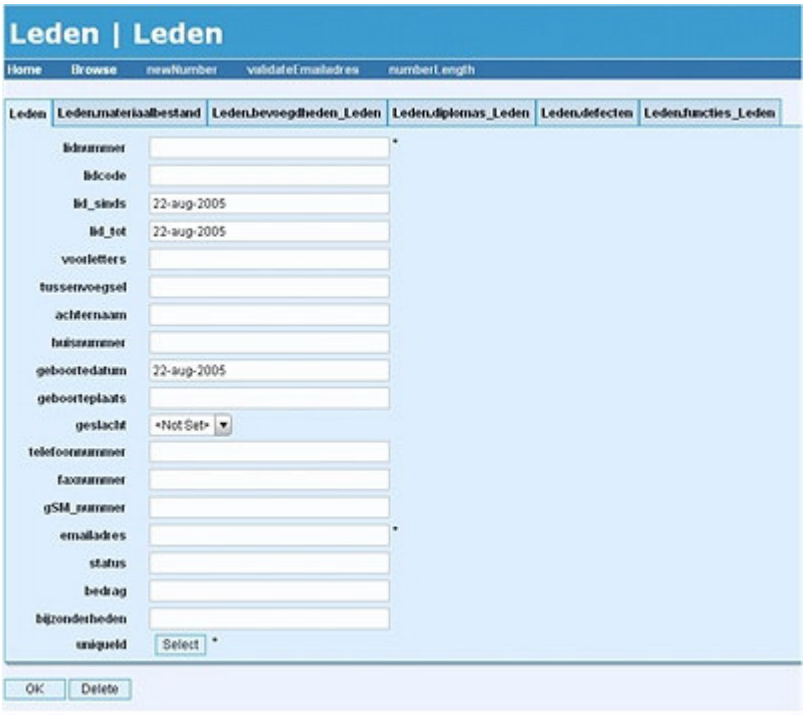
Een goed voorbeeld hiervan is te zien in figuur 5. Bij het testen van deze bescherming, blijkt deze prima te werken. De gegenereerde code is niet aan te passen en het programmeerde code wordt behouden bij het hergenereren van de code. Bij het beter bekijken van de code zien we dat er verschillende *import* statements staan, waarbij de programmeur indien nodig ook imports erbij kan plaatsen. Tevens zien we dat OptimalJ zelfs *If* statements heeft gegenereerd en de code van duidelijk commentaar heeft voorzien, zie figuur 6.

We kunnen zeggen dat OptimalJ meer dan alleen een raamwerk van code genereert en dat op de kwaliteit van de code niets aan te merken is. Wel met de opmerking erbij dat het niet genereren van zelfs de simpelste logica een groot gemis en extra werk voor de programmeur is.

5.5 Kwaliteit van de gegenereerde applicatie

Naast het beoordelen van de kwaliteit van de code hebben we ook gekeken naar de kwaliteit van de uiteindelijk gegenereerde applicatie. Het is natuurlijk belangrijk of de applicatie duidelijk, overzichtelijk en functioneel correct is.

Voor het bekijken en testen van de gegenereerde applicatie is de applicatie- en databaseserver van OptimalJ gebruikt. Zo werd de applicatie operationeel, waarvan het resultaat te zien is in figuur 7. Als we naar de applicatie kijken zien we alle velden die gemodelleerd zijn terug in de applicatie. Net als de originele applicatie heeft deze diversen tabbladen welk de bevoegdheden, diploma's en functies van het lid aangeven. Helaas staan er ook tabbladen tussen zoals materiaalbestand, defecten, etc. welke hier niet tussen horen. Dit kan een modelleringfout zijn of de gemiste correctie in het PSM, welke voor dit onderzoek niet is uitgevoerd. Als we gaan testen of een lid ingevoerd kan worden, blijkt dit enige problemen te geven. Er staan geen adressen in de database en wanneer we deze in willen voeren worden deze niet opgeslagen. De oorzaak van dit probleem kon niet gemakkelijk achterhaald worden en is verder ook niet onderzocht. Dit maakt het ook niet mogelijk een lid in te voeren omdat elk lid een adres moet hebben. We kunnen wel zeggen dat de restricties van de applicatie correct werken. Wat in de applicatie ook gemist wordt zijn duidelijke foutboodschappen. Foutboodschappen zoals bijvoorbeeld het verkeerd invullen van een datum. Wanneer men een datum verkeerd invoert wordt er alleen aangegeven dat er iets mis is met het datum veld maar verder niets.



Figuur 7: Applicatie gegenereerd door OptimalJ

We kunnen concluderen dat de gegenereerde applicatie een goed begin is, maar zeker nog niet klaar is om door de klant te laten gebruiken. De eerder genoemde fouten kunnen zijn voortgekomen uit het feit dat voor dit onderzoek het PSM niet is aangepast. Een mogelijkheid is ook dat we niet de juiste kennis hadden voor het correct ontwikkelen van een applicatie met OptimalJ. Hoe dan ook is het resultaat beter dan verwacht en is er bewezen dat een applicatie uit een model gegenereerd kan worden.

5.6 Tijd versus resultaat

Als laatste kijken we nog naar de verstreken tijd tegenover het eindresultaat. Dit is belangrijk omdat indien een goede programmeur sneller en beter een applicatie kan ontwikkelen dan een MDA tool, het voor de bedrijvenmarkt zinloos is te investeren in dit soort technologie.

Het ontwikkelen van software is afhankelijk van expertise van de ontwikkelaar, de grootte en de complexiteit van het project. Als we de ontwikkeltijd van de MDA tool OptimalJ zouden inschatten, bij het ontwikkelen van casus 3, dan zou dat ongeveer twee maanden zijn.
Uit eigen ervaring zal een programmeur die het zonder een MDA tool de applicatie moet programmeren twee tot drie maanden bezig zijn het gewenste resultaat voor de klant te behalen. Vanuit deze benadering kunnen we zeggen dat MDA iets sneller een applicatie kan leveren. Hierbij is de applicatie mogelijk ook beter gedocumenteerd. De programmeur is mogelijk langer met de applicatie bezig, maar kan deze meer naar wens van de klant programmeren.
Een ander voordeel van MDA is dat men met minder moeite en tijd de applicatie in orde kan maken voor een andere architectuur. Iets waar de programmeur waarschijnlijk meer moeite een tijd in moet steken. Mogelijk een onderwerp om meer onderzoek naar te doen.

Gezien het resultaat en de tijd die nodig is voor het ontwikkelen van een applicatie in OptimalJ, gaan de voorkeur nog steeds uit naar het ouderwets programmeerwerk. Dit komt doordat de gegenereerde applicatie niet naar behoren werkt. Wel is de verwachting dat na verdere ontwikkeling van dit soort MDA tools het in de toekomst mogelijk wordt betere applicaties te genereren.

6 Conclusies van het onderzoek

Het onderzoek richtte zich op de vraag: "Wat is de softwarekwaliteit van MDA tools?". Gezien de korte onderzoeksperiode is er gekozen voor het onderzoeken van één MDA tool, namelijk OptimalJ van Compuware. Ten behoeve van het onderzoek is er een duidelijke definitie en uitleg van MDA neergezet waarna vastgelegd kon worden of OptimalJ een MDA tool is. Hiervoor moest eerst gekeken worden of de verschillende aspecten van de voorgeschreven MDA standaards in de tool voorkomen. Geconcludeerd is dat, desondanks andere benamingen, het PIM, PSM en code model wel degelijk in OptimalJ voorkomen. Dit geldt ook voor de transformaties tussen de modellen en de mogelijkheid tot het kiezen van een architectuur en specifieke platform. Dit laatste is alleen haalbaar wanneer men gebruik maakt van de Architect Edition en een pattern voor het andere platform maakt. Standaard ondersteund OptimalJ Professional Edition namelijk het J2EE platform.

Opvallend is de veelzijdigheid van de tool, met mogelijkheden zoals: testen, debuggen, bescherming van zelfgeschreven en gegenereerde code, ingebouwde applicatieserver, ondersteuning voor andere ontwikkelomgevingen en de databaseserver SOLID. Dit alles is verwerkt in een duidelijke en overzichtelijke ontwikkelomgeving van OptimalJ zelf. Een ander punt is de controle van de minimale systeemeisen, zoals het fysieke geheugen van minimaal 512MB. Dit draagt bij aan een stabiele en goedwerkende MDA tool.

Om de softwarekwaliteit te onderzoeken zijn er onderzoeksvragen van te voren opgesteld. Dit helpt antwoord te geven op de hoofdvraag. Eén van de onderzoeksvragen is hoe de tool omgaat met de schaalbaarheid van het model. Na het uitvoeren van verschillende casussen en het bestuderen van de genoteerde geheugenwaarden kan er geconcludeerd worden dat er een lineair verband is tussen de grootte van het model en hoogte van het geheugen. Een opmerking hierbij is dat bij het heruitvoeren van de casussen niet dezelfde geheugenwaarden naar voren kwamen, maar wel een lineaire stijging van het geheugen voorkwam.
Ook is onderzocht wat de prestaties van de MDA tool zijn. Hiervoor zijn tijdens het uitvoeren van de casussen de tijden, hoelang OptimalJ over de transformaties doet, geregistreerd. Deze tijdregistratie is in een grafiek, gecategoriseerd per transformatie, verwerkt. De grafiek geeft een goed beeld hoe de tijd tegenover de grootte van een model verloopt. Eveneens is hier per transformatie een lineair verband te herkennen, indien men vanaf de beginwaarde een rechte lijn trekt naar de laatste waarde zijn er wel enkele lichte afwijkingen te herkennen. Het doet niet af aan de verwachting hoelang OptimalJ er over doet om 10 klassen met onderlinge relaties en datatypes te transformeren naar het volgende abstractieniveau. Volgens de grafiek zou dit, voor de transformatie van het PIM naar het PSM, ongeveer 1 minuut en 26 seconden zijn. Andersom kan dit ook gedaan worden om te weten te komen hoe groot het model zou zijn bij bijvoorbeeld het transformeren van het PSM naar de platform specifieke code. Als we de lijn doortrekken naar 5 minuten, dan komen we op ongeveer 38 klassen uit. Dit geeft aan dat het uitvoeren van de transformaties geen enorme projectvertragingen zullen veroorzaken. Bij het onderzoeken van de prestaties bij andere MDA tools kan men meer zeggen over hoe OptimalJ presteert ten opzichte van de anderen.
Het onderzoek zou onvolledig zijn als er niet gekeken werd naar de kwaliteit van de gegenereerde code en gecompileerde applicatie. De bescherming van de geschreven en de gegenereerde code is hierbij getest en geslaagd. De programmeur hoeft zich geen zorgen te maken dat de geschreven code, na het opnieuw uitvoeren van het MDA proces, overschreven zou worden door de transformaties. Een gemis is het niet kunnen genereren van simpele logica, zoals het bij elkaar optellen of samenvoegen van velden. Hierdoor moet de programmeur dit in het code model nog erbij programmeren. Bij het evalueren van de applicatie bleek deze niet naar behoren te functioneren, de oorzaak hiervan is niet achterhaald. Mogelijk een fout in het proces van modelleren of het niet correct werken van de database-engine. Vooralsnog bestaat de applicatie uit een duidelijke structuur, lay-out en navigatie, wat middels het modelleren van het PSM aangepast kan worden.

Afsluitend kan er geconcludeerd worden dat de softwarekwaliteit van de MDA tools in het algemeen afhankelijk is van expertise van de ontwikkelaar(s), grootte en complexiteit van de toekomstige applicatie. De softwarekwaliteit van OptimalJ is redelijk. Er is nog steeds een lange weg te gaan om MDA of MD(S)D als standaard van de Nederlandse softwareontwikkelingmarkt te krijgen.

Literatuurlijst

Boeken

[Kleppe 03] Kleppe A., Warmer J. en Bast W. (2003) *MDA Explained: The Model Driven Architecture: Practise and Promise*, Addison-Wesley – Pearson Education Inc., ISBN 0-321-19442-X

[Raistrick 04] Raistrick C. et al. (2004) *Model Driven Architecture with Executable UML*, Cambridge, ISBN 0-521-53771-1

[Rumbaugh 99] Rumbaugh J., Jacobson I. en Booch G. (1999) *The Unified Modeling Language Reference Manual*, Addison-Wesley, ISBN 0-201-30998-X

Papers

[Belo 04] Belo W. (2004) *Model Driven Architecture: Specificeren van software functionaliteit*, Universiteit van Amsterdam, juni 2004

[Brand 04] Brand M. van der (2004) *Softwarekwaliteit – Hypes versus onderzoek in de software engineering*, Hogeschool van Amsterdam, 15 december 2004, http://www.hva.nl/lectoraten/ol07-041215-vandenbrand.pdf

[Czarnecki 03] Czarnecki K. en Helsen S. (2003) *Classification of Model Transformation Approaches*, University of Waterloo, Canada, http://www.swen.uwaterloo.ca/~kczarnec/ECE750T7/ czarnecki_helsen.pdf

[Gardner 02] Gardner et al. (2002) *A review of OMG MOF 2.0 Query / Views / Transformations Submissions and Recommendations towards the final standard*, http://www.zurich.ibm.com/pdf/ ebizz/gardner-etal.pdf

[Greenfield 03] Greenfield J., Short K. (2003) *Software Factories: Assembling Applications with Patterns, Models, Frameworks and Tools*, California - USA, 26 Oktober 2003

[Meijles 05] Meijles J. (2005) *Ontwikkelen met MDA voor Dummies*, Universiteit van Amsterdam

[Middleware_1 03] The Middleware Company (2003) *Model-Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach*, Juni 2003, http://www.middlewareresearch.com

[Middleware_2 03] The Middleware Company (2003) *Model-Driven Development for J2EE with IBM Rational Rapid Developer (RRD)*, November 2003, http://www.middlewareresearch.com

[Middleware_3 03] The Middleware Company (2003) *Model-Driven J2EE Development: Comparing Two Approaches and Tools*, November 2003, http://www.middlewareresearch.com

[Mohag 03] Mohagheghi P., Pettersen J., Najib S. en Najib W. (2003) *MDA and Integration of Legacy Systems: An Industrial Case Study*, Ericson Noorwegen – Grimstad / Department of Computer and Information Science / Simula Research Laboratory / Agder University College , Noorwegen juni 2003, http://www.idi.ntnu.no/grupper/su/publ/pdf/mda-paper-19jun2003.pdf

[Najib 03] Najib S. en Najib W. (2003) *MDA and Integration of Legacy Systems*, Agder University College, Noorwegen mei 2003, http://student.grm.hia.no/master/ ikt03/ikt6400/g04/ SeloWarsun.pdf

[Runde 03] Runde R.K., Stolen K. (2003) *What is Model Driven Architecture?,* Universiteit van Oslo, ISBN 82-7368-256-0, ISSN 0806-3036, http://heim.ifi.uio.no/~ketils/sardas/UIO-IFI-RR304.pdf

[Witkop 03] Witkop S. (2003) *Driving business agility with Model Driven Architecture*, EDS, Ohio, Pennsylvania Solution Centre, http://eds.com

[Völter 04] Völter M. en Bettin J. (2004) *Patterns for Model-Driven Software-Development*, Völter – ingenieursbureau voor softwaretechnologie, Duitsland / SoftMetaWare, New Zeeland, http://www.voelter.de/data/pub/MDDPatterns.pdf

Documentatie & Specificatie

[IEEE 02] IEEE (2000) *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems*, IEEE Std 1471-2000, ISBN 0-7381-2519-9

[MDA] OMG (2003) *The Model-Driven Architecture*, Guide Version 1.0.1, OMG Document: omg/2003-06-01

[OptimalJ] Compuware (2005) *OptimalJ 3.0*, User's Guide, http://www.compuware.com/products/optimalj

[UML] OMG (2003) *OMG Unified Modeling Language Specification*, versie 1.5, Maart 2003, OMG Document: formal/03-03-01

Websites

[AndroMDA] AndroMDA een open source product, http://www.andromda.org/index.html

[Compuware] Compuware over OptimalJ, http://www.compuware.com/products/optimalj/default.htm, geraadpleegd op 31 juni 2005

[IBM] IBM's Rational Rose XDE developer, http://www-306.ibm.com/software/awdtools/developer/rosexde/

[IO] Interactive Objects met hun product ArcStyler, http://www.arcstyler.com/

[MDSD] MDSD over Model Driven Software Development, http://www.mdsd.info, geraadpleegd op 10 juli 2005

[OAW] openArchitectureWare Tool, http://architecturware.sourceforge.net/

[OMG_MDA] OMG over MDA, http://www.omg.org/mda, geraadpleegd op 12 mei 2005

[OMG_UML] OMG over UML, http://www.uml.org, geraadpleegd op 20 juni 2005

Bijlage A: Casussen t.b.v. onderzoek

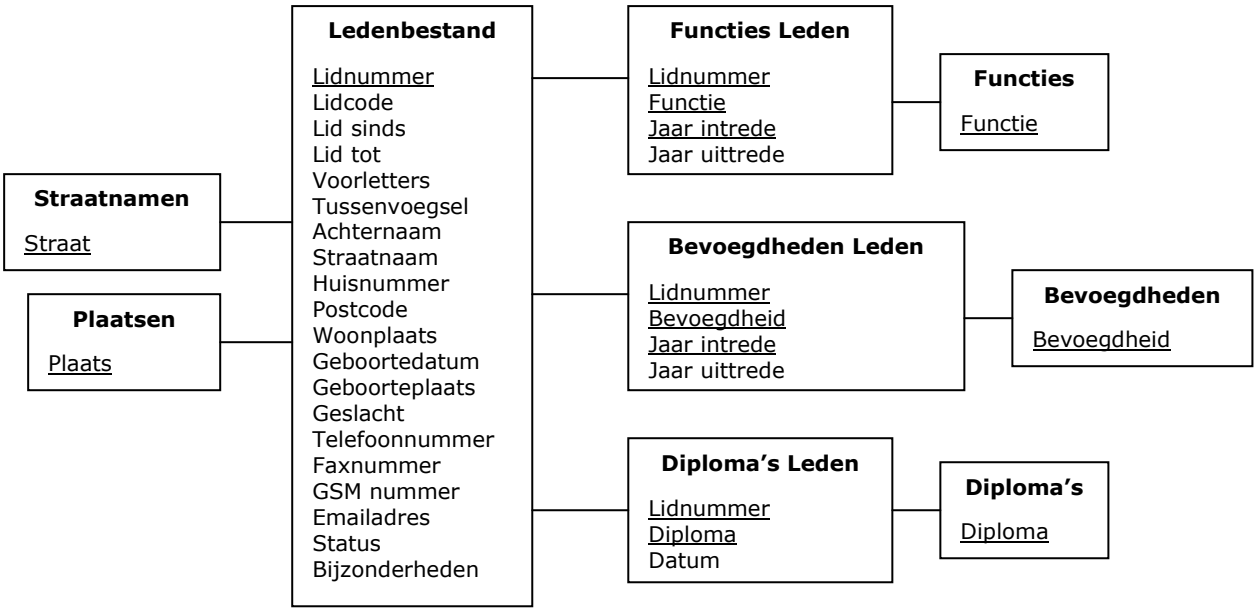
De casussen die in dit onderzoek uitgevoerd zijn hebben betrekking op de schaalbaarheid van de MDA modellen. De casussen zijn gebaseerd op een reeds ontwikkeld en geïmplementeerd systeem van de IJmuider Reddingsbrigade (IJRB). Deze organisatie heeft door de auteur een administratieprogramma laten ontwikkelen. Dit programma heeft als architectuur een presentatie-, business en datalaag, met andere woorden een drielagen architectuur. Elke casus bevat een uitbreiding van het systeem, om zodoende de schaalbaarheid te onderzoeken.

Casus 1 – Ledenadministratie

Deze casus beschrijft de ledenadministratie van de IJmuider Reddingsbrigade. Het beveiligings-aspect van de gegevens is niet meegenomen omdat dit niet relevant is voor de schaalbaarheid.

Van alle leden moeten hun naam, adres, geboorte- en lidgegevens geregistreerd worden. Hierbij worden allerlei extra gegevens ook geregistreerd, zoals: diploma's, functies binnen de vereniging en bevoegdheden bij het reddingswerk. Er zijn ook een aantal velden waarbij het systeem berekeningen zal uitvoeren. Velden zoals: leeftijd, aantal jaar lid en nieuw lidnummer. Tevens zullen op diversen velden controles worden uitgevoerd om de gegevens te valideren. De velden zijn: lidcode, postcode, telefoonnummer, emailadres en datumvelden.

Dit alles kan worden afgelezen in het volgende figuur:



Figuur 8: Datamodel ledenadministratie van IJRB database

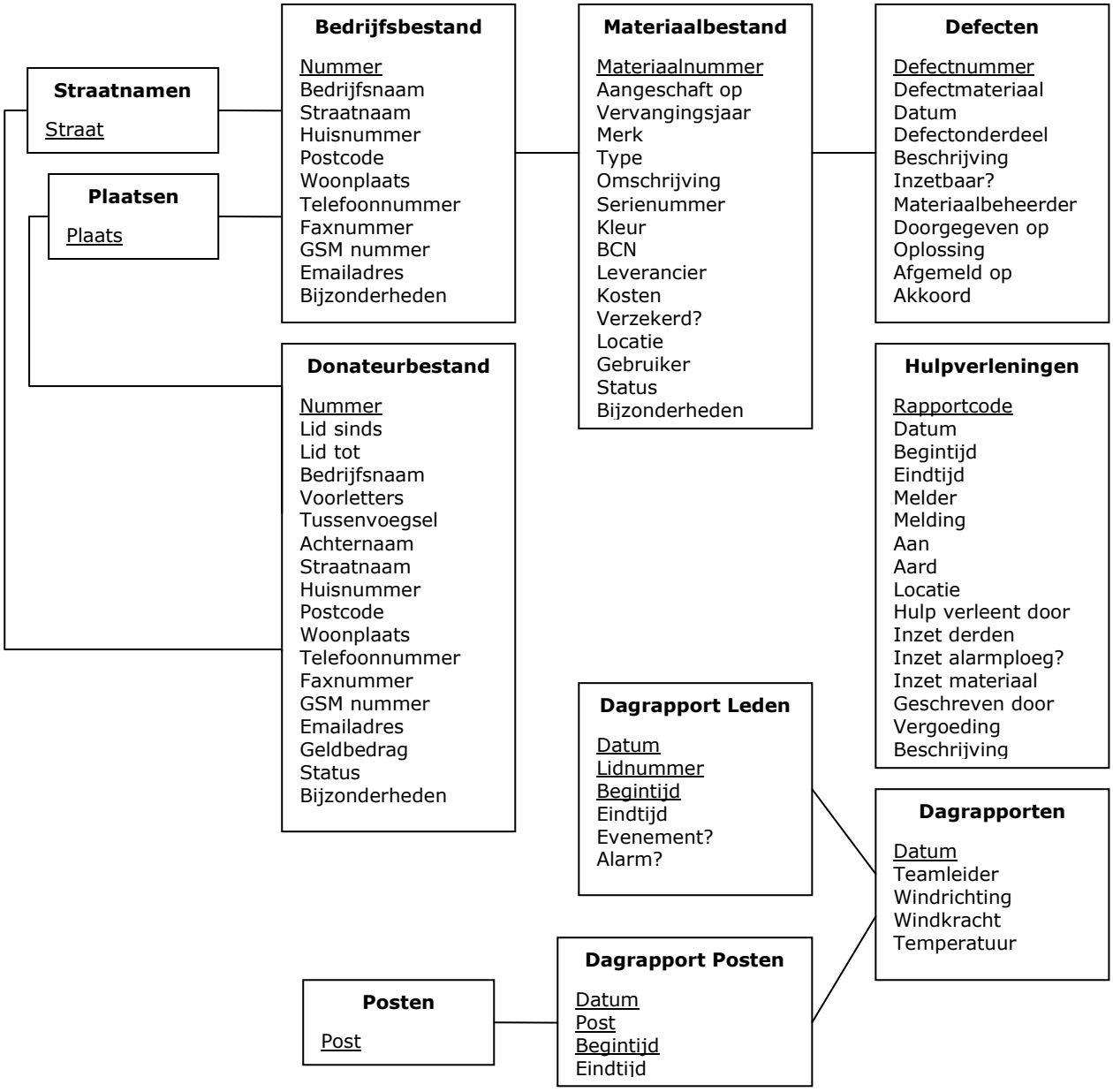
Casus 2 – Uitbreiding met Strandzaken

Deze casus is een uitbreiding van de casus 1, hierbij worden de modellen uitgebreid met een business specifieke administratie.

Onder een business specifieke administratie wordt verstaan:

- Administreren van donateurs/sponsoren;
- Administreren van bedrijven/leveranciers voor het materiaal;
- Bijhouden van de uren die de strandploeg maakt per dag;
- Bijhouden van de diverse hulpverleningen die verleend zijn;
- Bijhouden van het materiaal dat de IJRB bezit;
- Bijhouden van de defecten aan het materiaal.

De bovengenoemde administratietaken worden in het volgende figuur weergegeven.



Figuur 9: Datamodel overige functies van IJRB database

Casus 3 – Uitbreiding met Statistieken

Deze casus is een uitbreiding van de casus 2, een extra aandachtspunt hierbij is het stukje logica dat er gegenereerd moet worden.

Het systeem wordt met deze casus uitgebreid met statistieken. Hierbij zal voornamelijk op de businesslaag een grote uitbreiding plaatsvinden. Er zullen diversen classes gegenereerd moeten worden voor het berekenen van diversen waarden ten behoeve van de organisatie. Enkele voorbeelden van statistieken zijn:

- Totaal aantal uren strandploegleden (per jaar en maand);
- Totaal aantal hulpverleningen (per jaar en maand);
- Totaal aantal alarmeringen (per jaar en maand);
- Totaal aantal defecten (per jaar en maand);
- Het aantal reddingen (per jaar en maand);
- Het aantal reanimaties (per jaar);
- Uren per strandlid met aantal keer en gemiddelde over een jaar;
- Totaal aantal donateurs/sponsoren;
- Totaal aantal leden;
- Leden per status + aantal;
- Leden die lid zijn geworden dit jaar + aantal;
- Leden die zijn gestopt dit jaar + aantal.

Bijlage B: Plan van Aanpak

Inleiding

Dit is het plan van aanpak van de afstudeerperiode van Edwin de Groot bij het CWI betreffende het onderwerp "Model Driven Architecture". Het afstudeerproject zal 3 maanden duren, van 1 april tot 1 juli 2005. De student zal worden begeleid door Mark van den Brand, Lector UvA en onderzoeker CWI. In dit plan van aanpak zijn de projectomschrijving, verwachte resultaten, projectexpertise, risico's, activiteiten, planning en literatuurlijst benoemd.

Projectomschrijving

Het onderzoek betreft het analyseren de kwaliteit van MDA tools. Er zijn verschillende MDA transformaties tools te vinden op internet. Maar in hoeverre kunnen zij grote UML transformaties aan en beloven zij het gewenste resultaat. Aangezien de scope van dit project zeer breed is, richt ik mij speciaal op één MDA tool en ga deze op verschillende kwaliteitsattributen testen. Dit gebeurt door middel van het opstellen van casussen en deze met de tool uit te voeren. De tool die ik voor dit project ga gebruiken is OptimalJ.

Mijn onderzoeksvraag is: "Hoe staat het met de softwarekwaliteit van MDA tools?"

Om dit project tot een goed einde te brengen heb ik de onderzoeksvraag verdeeld over meerdere kleine onderzoeksvragen:

- Is de MDA tool gemakkelijk te hanteren of kost het de gebruiker enige moeite om een simpele model transformatie uit te voeren?
- Wat is de tijdsduur van de transformatie? En hoe staat deze in verhouding met het resultaat?
- Kan een MDA tool een model ter grootte van een SAP-pakket genereren en daaruit een daadwerkelijk pakket transformeren?
- Hebben MDA tools modelleringgrenzen of is dit verbonden aan de hardware van de PC?

Elke grootte van casus zal ik uitvoerig bespreken in mijn scriptie en wat mijn bevindingen hiervan zijn.

Verwachte resultaten

Ik verwacht van een MDA tool, zoals OptimalJ, dat deze een groot softwarepakket moet kunnen genereren. Persoonlijk denk ik niet dat er een werkend programma uit zal komen omdat de techniek nog in de kinderschoenen staat en met de huidige softwarekennis het nog onmogelijk is een werkend programma te genereren. Vandaar dit onderzoek of de kwaliteit van de transformaties dat is wat beweert wordt. Hierbij is een belangrijk aspect meegenomen, namelijk de tijd tegenover het resultaat.

Projectexpertise

De volgende expertises zijn nodig voor het correct afronden van dit project:

- Java programming
- UML designing
- MDA knowledge
- Kennis van softwarekwaliteit

Risico's

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

Risicofactor	Kans	Schade	Risico	Maatregels / bijzonderheden
Werkplek	-	++	+	Mogelijk bij CWI/UvA en thuis
Tijdsduur stage	-	-+	-+	Harder werken Langer doorgaan met stage
Documentatie	++	+	+	Vroegtijdig verzamelen Externe kennis er bij halen
Licenties	+	++	+	Backup besturingssysteem en voor elke casus deze terug zetten
Beschikbaarheid begeleiding	+	-	-+	Dit is een zelfstandig onderzoek en de leerling dient met minimale hulp dit te volbrengen.
Kennis	+	+	-+	Kennis is te vergaren door lezen van literatuur en extra tijd hierin te steken ten behoeve van afstuderen
Beschikbaarheid Specs	++	+	+	Dit zou een probleem kunnen zijn, hiervoor zouden we aannames en van mijn eigen bevindingen uit moeten gaan.

Activiteiten

- Scriptie schrijven en bijhouden van resultaten hierin;
- Kennis maken met de materie en het werkgebied;
- Onderzoeken en documenteren bevindingen;
- Casus opstellen;
- Casus UML model transformatie van PIM naar PSM naar code uitvoeren;
- Analyse resultaten;
- "Tussentijds"scriptie schrijven over bevindingen en resultaten afgelopen onderzoek;
- Mogelijkheid voor nog een ronde voor een groter UML model, hierbij herhalen van laatste 4 activiteiten.

Planning

De planning ziet er als volgt uit: