Scriptie Master Software Engineering

Datatransformaties door middel van Model Driven Architecture.

UNIVERSITEIT VAN AMSTERDAM

Student:	Bart Vreeken
Opleiding:	Master Software Engineering Universiteit van Amsterdam
Docent:	Hans Dekkers
Stagebegeleider:	Etienne Bido
Opdrachtgever:	Ordina J-technologies

Datum:	31 Augustus 2006
Status:	Final
Versie:	1.3

Inhoudsopgave

Samenvatting

Het document beschrijft de uitvoering van de afstudeeropdracht. In drie maanden tijd is onderzocht of de MDA tool AndroMDA geschikt is om datatransformaties uit te voeren voor Ordina. De aanleiding voor dit onderzoek is enerzijds de behoefte van Ordina om op zoek te gaan naar efficiëntere ontwikkelmethodes. Anderzijds is er de behoefte het programmeren van XSLT te vereenvoudigen, dat wordt gebruikt voor de datatransformaties bij service georiënteerde integratie van verschillende systemen.

- Als efficiënte ontwikkelmethode is gekozen voor de relatief nieuwe technologie *Model Driven Architecture* (MDA). MDA beschrijft een methode waarbij de software ontwikkeling plaatsvindt d.m.v. UML modellen. Voor de ondersteuning van deze methode is gekozen voor de open-source tool AndroMDA. Deze tool is in staat om vanuit een platform onafhankelijk UML model programmacode te genereren voor een specifiek platform (zoals Java of .Net).
- XSLT is een taal die XML documenten kan transformeren naar elk ander, op tekst gebaseerd, formaat. In dit onderzoek is XSLT gebruikt om een XML document te transformeren naar een ander XML document. XSLT is dus verantwoordelijk voor het uitvoeren van de datatransformatie.

Beide behoeftes zijn gecombineerd en zodoende is er een experiment opgesteld waarin is onderzocht of de MDA tool AndroMDA bruikbaar is om XSLT efficiënt te ontwikkelen.

Tijdens het experiment is er bepaald welke datatransformaties de XSLT template moet kunnen uitvoeren. Dit zijn de criteria van de klant. Vervolgens is er een plugin voor AndroMDA ontwikkeld - een *AndroMDA cartridge* – die in staat is vanuit een UML diagram een XSLT template te genereren. De ontwikkeling van de cartridge speelt een centrale rol in het experiment en bevat o.a. de transformatie definitie om het UML diagram om te zetten naar XSLT. Alhoewel deze definitie eerst in natuurlijke taal wordt uitgedrukt om te beredeneren hoe de omzetting moet plaatsvinden, is deze als complexe programmeercode geïmplementeerd in de cartridge, samen met verschillende configuratie bestanden.
Tevens is er gedurende het experiment een theoretische studie gedaan naar datatransformaties in het algemeen. Dit leverde – naast de reeds gevonden criteria van de klant – een lijst op met typen datatransformaties die mogelijk zijn. Er zijn grofweg 18 typen transformaties in de categorieën ‘structuur’, ‘string’, ‘datum/tijd’, ‘wiskundig’ en ‘conditioneel’ die geïmplementeerd zouden moeten worden in een AndroMDA cartridge om volledige datatransformatie functionaliteit te bieden.

Het experiment heeft uiteindelijk inzicht gegeven in de mogelijkheden van AndroMDA, om te beredeneren welke typen datatransformaties nog meer mogelijk zouden zijn met deze tool. Ook zijn er conclusies getrokken wat betreft de toepasbaarheid van AndroMDA en de MDA concepten in andere projecten van Ordina.
In de opgeleverde cartridge zijn niet alle criteria van de klant geïmplementeerd. In de praktijk is er een aantal knelpunten, zoals de beperkte uitdrukkingskracht van het UML model om datatransformaties te modelleren. Ook de transformatie definitie in de cartridge wordt een onbeheersbare brij van programmacode wanneer er teveel typen datatransformaties worden geïmplementeerd. Het laatste knelpunt is vooral een gevolg van de complexe syntax van XSLT, waardoor geconcludeerd moet worden dat het uitgevoerde experiment (en datatransformaties in het algemeen) niet bijzonder geschikt is voor AndroMDA.
Echter, ondanks de ongeschiktheid voor datatransformaties heeft de MDA tool AndroMDA genoeg te bieden voor Ordina. De tool heeft voldoende ondersteuning voor het type projecten die Ordina uitvoert en biedt naar mijn inziens de voordelen die MDA te bieden heeft. AndroMDA kan ontwikkelaars dwingen te werken onder architectuur, levert een hogere productiviteit en heeft voldoende flexibiliteit om gegenereerde code uit een UML model aan te passen aan de wensen van de klant.

Voorwoord

Dit document beschrijft het afstudeerproject van Bart Vreeken. Het project is onderdeel van de laatste fase van de opleiding Master Software Engineering aan de Universiteit van Amsterdam en is uitgevoerd in opdracht van Ordina J-technologies, een afdeling binnen Ordina N.V.

Het doel van dit document is in eerste instantie aan te tonen dat ik in staat ben een wetenschappelijk onderzoek op gestructureerde wijze uit te voeren én uiteen te zetten. Tevens hoop ik hiermee mensen (of meer specifiek: software ontwikkelaars) bij Ordina te interesseren voor het onderwerp uit mijn onderzoek.

Het document is als volgt opgebouwd. Het eerste hoofdstuk bevat alle inleidende informatie. Naast dat het de probleemstelling en onderzoeksvraag formuleert, bevat het ook achtergrondinformatie over de belangrijkste technologieën die zijn toegepast in de rest van het document. Voor de lezers die niet bekend zijn met de concepten van MDA en UML is dit verplichte kost. Hoofdstuk twee beschrijft de gefaseerde aanpak die is gevolgd tijdens het project, waarna in hoofdstuk drie de theorie van datatransformaties wordt behandeld. Hoofdstuk vier beschrijft de case die is uitgevoerd. Dit technische hoofdstuk beschrijft de architectuur van de MDA tool 'AndroMDA' en de oplossing die is gerealiseerd naar aanleiding van de probleemstelling: Een software oplossing voor AndroMDA. Aan de hand van de theorie over datatransformaties uit hoofdstuk drie en de inzichten die zijn verworven door het uitvoeren van de case, worden er in hoofdstuk vijf conclusies getrokken betreffende de onderzoeksvraag. Tevens wordt ingegaan op de toepasbaarheid van AndroMDA en de MDA concepten bij Ordina. Het laatste hoofdstuk besluit met het geven van de resultaten.

Als laatste rest mij nog te zeggen dat ik drie maanden lang ontzettend veel plezier heb gehad bij Ordina, en daarnaast ook erg hard heb gewerkt om alles op tijd af te ronden. Ik wil de volgende mensen bedanken: Mijn begeleider bij Ordina: Etienne Bido. Jos Warmer, Hans Kuijpers, Willem van Pruijssen, Kasper Helmers en anderen voor hun technische begeleiding. En van de UvA mijn begeleider Hans Dekkers, voor zijn kritische feedback.

Veel plezier met het lezen van mijn scriptie.

Bart Vreeken
Augustus 2006, Amersfoort.

1. Inleiding

1.1. Aanleiding

Een algemene trend in de software industrie is dat nieuwe ontwikkelingen in software met argusogen worden gade geslagen. Dat zien we bijvoorbeeld nu, op het moment van schrijven: nieuwe ontwikkelingen doen zich voor op het gebied van internet technologie waarbij de grenzen tussen web- en desktop applicaties vervagen [Ct200605], bekend als Web 2.0. Dit roept bij een aantal ontwikkelaars herinneringen op aan het mislukte concept van Oracle's *Network Computer* (NC), waarbij de domme - maar goedkope - terminal gegevens en programma's van het net moest halen.
Ook een relatief nieuwe technologie is *Model Driven Architecture* (MDA). In 2001 lanceerde de *Object Management Group* (OMG) het MDA initiatief waarbij het ambitieuze doel werd gesteld om de focus van software ontwikkeling te verschuiven van het schrijven van code naar het UML modelleren, hetgeen een hoger abstractie niveau van ontwikkelen betekent (hierover meer in het volgende hoofdstuk). Ook deze nieuwe technologie roept bij ontwikkelaars gemengde gevoelens op. Aan de ene kant is er de behoefte om op een hoger abstractie niveau software te ontwikkelen. Een hoger abstractie niveau kan o.a. leiden tot het efficiënter ontwikkelen van software, wat een concurrentie voordeel oplevert. We zien dat bijvoorbeeld terug in de overstap die is gemaakt van assembly taal naar een 'high-level' programmeertaal als Java.
Aan de andere kant staat een eerdere poging uit de software industrie om op een hoger abstractie niveau te ontwikkelen in een vierde generatie taal, in de vorm van *Computer-Aided Software Engineering* (CASE) tools. Echter, CASE tools zijn vrijwel fabrikant specifiek en er wordt geen gebruik gemaakt van standaard specificaties, zoals nu het geval is bij MDA.

Nu het MDA framework meer volwassen is geworden (uitbreiding van specificaties) en een betere tool ondersteuning kent, zijn bedrijven langzamerhand bereid te onderzoeken wat deze nieuwe manier van ontwikkelen kan opleveren. Ook het bedrijf *Ordina* - waar dit afstudeeronderzoek plaats vindt – is geïnteresseerd in de mogelijkheden en de beloftes rond MDA. Ordina is één van de grootste beursgenoteerde IT dienstverleners van Nederland, en bedient met zijn drie kernactiviteiten consulting, ICT en outsourcing de 'top40' grootste bedrijven in Nederland. Ordina is (net als vele andere IT bedrijven) op zoek naar innovatieve manieren om te kunnen concurreren met IT bedrijven uit lage lonen landen, zoals India.
Bij Ordina is er een klein aantal projecten uitgevoerd waarbij de MDA tool *OptimalJ* van Compuware is gebruikt om *alle* software 'artifacts' mee te genereren vanuit UML modellen (denk aan grafische schil, database schema's, etc). Wat opvalt is dat in onderzoeken naar MDA [Belo2004, Thomas2004] de nadruk ligt op wat er *niet* met MDA tools mogelijk is.

In dit afstudeeronderzoek ligt de nadruk op wat de concepten rondom MDA én ondersteunende tools nú voor bedrijven kan betekenen. Immers, wanneer het gebruik van MDA tools al een beperkte tijdwinst kan opleveren is dit interessant (afgezien van opleidingskosten, licentiekosten, etc). Dit onderzoek kent een aantal interessante invalshoeken:

- Er wordt gebruik gemaakt van open-source tools.
  In de MDA projecten bij Ordina is er tot op heden alleen gebruik gemaakt van het commerciële MDA pakket OptimalJ. Nieuwe MDA projecten zijn niet van de grond gekomen omdat dit o.a. hoge licentiekosten met zich meebrengt voor de aanschaf van tools.
  In dit afstudeeronderzoek wordt gebruik gemaakt van de open-source MDA tool *AndroMDA*, waar binnen Ordina nog geen kennis van in huis is. Eerste vooronderzoeken hebben uitgewezen dat AndroMDA dezelfde mogelijkheden kan bieden als *OptimalJ* (fabrikant Compuware) en op een aantal fronten meer flexibiliteit biedt dan de commerciële concurrent (implementatie van een andere template engine, implementatie van een andere repository engine om een andere modelleertaal te ondersteunen dan het *Model Object Facility* (MOF) model).

- MDA tooling wordt ingezet als *code generator* in plaats van *applicatie generator*.

Zoals eerder genoemd zijn MDA tools in vorige projecten gebruikt om alle software artifacts in het project te genereren. Zo levert het genereren van de database schema's meer succes op dan het genereren van de grafische vensters (hetgeen moeilijk te modelleren is in UML). In plaats van te onderzoeken hoe de MDA tool AndroMDA een applicatie in zijn geheel kan realiseren, wordt er gekeken op welke fronten AndroMDA goed dienst kan doen als code generator. AndroMDA wordt op deze wijze ingezet op de punten waar het in uitblinkt.

Om AndroMDA als code generator te onderzoeken is er een specifiek platform gekozen die als case kan dienen om de bruikbaarheid van AndroMDA aan te tonen. Als case is er voor gekozen om code te genereren om datatransformaties uit te voeren: hetgeen wordt verduidelijkt in hoofdstuk 1.3.

1.2. Model Driven Architecture en UML

Zoals eerder beschreven wordt er tijdens deze afstudeeropdracht onderzoek gedaan naar *Model Driven Architecture* (MDA), ontwikkeld door de Object Management Group (OMG) in 2001. MDA biedt een framework waarin het ontwikkelen van software door middel van modellen centraal staat. Een 'key concept' in MDA is dan ook dat de focus verschuift van code schrijven naar 'modelleren', waarbij de modellen een beschrijving vormen van het te ontwikkelen systeem. Het 'programmeren' gebeurt dus niet met code, maar met modellen die formeel gedefinieerd zijn [Kleppe2003]. De formele taal van deze modellen is UML (tevens een specificatie van de OMG).
Naast dat software ontwikkeling met MDA 'model driven' is, doet de naam tevens vermoeden dat MDA iets met architectuur te maken heeft. Echter, dit is niet het geval. De beste verklaring voor deze naamkeuze is het feit dat het MDA ontwikkelaars in staat stelt software te ontwikkelen die aan een bepaalde architectuur voldoet (hetzij door dit zelf te modelleren, hetzij vastgelegd in transformatie regels).

Ontwikkelen van software vindt dus grotendeels plaats door het ontwikkelen van formele UML modellen. Dit houdt in dat de UML modellen door een tool zijn in te lezen en van daaruit worden getransformeerd naar programma code. Dit hoofdstuk geeft algemene informatie over MDA en UML en geeft een beter begrip over beiden, omdat zowel UML als MDA een belangrijke rol spelen in dit project. De paragrafen over MDA zetten uiteen uit welke onderdelen het MDA framework bestaat en hoe de MDA tool AndroMDA hier in past. De paragrafen 1.2.4. en 1.2.5. beschrijven hoe UML in allerlei specifieke domeinen bruikbaar is als modelleertaal, door middel van UML uitbreidingen. Lezers die reeds bekend zijn met MDA en UML kunnen deze paragrafen overslaan.

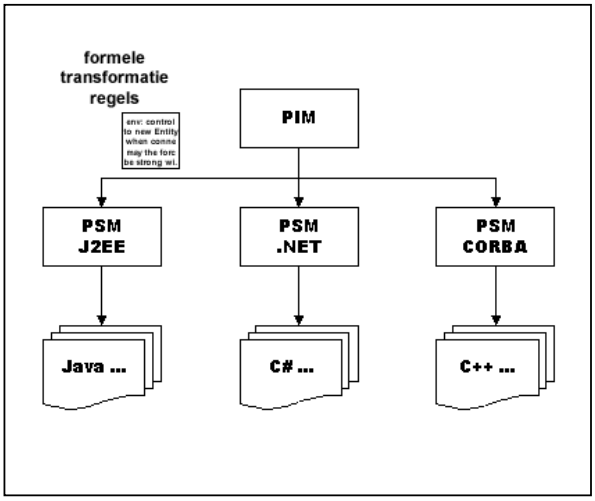
1.2.1. Het MDA framework

De OMG heeft met MDA een framework opgesteld waarin alle elementen zijn gedefinieerd die samen verantwoordelijk zijn voor het ontwikkelen van software met behulp van MDA. Het framework is globaal weergegeven in figuur 1 en bestaat uit de volgende elementen en toont tevens de stappen van model naar code:

Platform Independent Model – Het eerste model uit MDA is het Platform Independent Model (PIM)[1]. Het PIM beschrijft het software systeem op een abstractie niveau **in UML** waarbij het de business ondersteunt, zónder daarvan een technische implementatie te geven. Dit model is (in theorie) het enige model wat volledig 'met de hand' wordt ontwikkeld: verdere onderdelen worden automatisch gegenereerd door transformaties.

Platform Specific Model – In de volgende fase – wanneer een PIM ontwikkelaar alle elementen van de software op een platform onafhankelijk niveau heeft gemodelleerd – wordt het PIM getransformeerd naar het Platform Specific Modellen (PSM). Deze bevat alle platform specifieke details. Het woord platform dient men daarbij in de meest ruime zin van het woord te nemen. Een platform kan gaan over verschillende programmatalen (Java, C#) of platformen die complementair zijn binnen de software (SQL schema, Java EJB en JSP). Zoals in figuur 1 is te zien wordt een PIM getransformeerd naar één of meerdere PSM's. In de laatste stap wordt elke PSM omgezet naar programmacode.

[1] MDA kent ook het Computable Independent Model (CIM), een model wat voor het PIM komt waar alle bedrijfsregels in zijn opgenomen. Het CIM valt buiten de scope van het onderzoek.



Figuur 1 – *Het MDA framework.*

Transformaties – Transformaties vormen een essentieel onderdeel van het MDA framework. Een transformatiedefinitie zorgt er voor dat een model op het ene abstractie niveau (PIM) wordt vertaald naar een model op lager abstractie niveau (PSM). Om dit te bewerkstellen bevat zo'n transformatiedefinitie een set met transformatieregels die beschrijven hoe elementen uit het bron model vertaald worden naar elementen in het doel model. De MDA tool past de regels vervolgens toe waarna een volgend model (of programmacode) ontstaat. Omdat binnen het MDA framework uitsluitend transformaties plaatsvinden van een hoger naar een lager abstractie niveau (van PIM naar PSM), bevatten de transformatieregels platform specifieke details. Elk platform specifiek model bevat dus een bijbehorende set met transformatieregels.
Het gehele transformatie proces is in feite een black box voor ontwikkelaars die met MDA tools werken. De transformatiedefinities zijn veelal door de fabrikant opgesteld en worden als 'plugin' aangeboden aan de MDA ontwikkelaars om te gebruiken binnen de MDA tool. Het enige wat de ontwikkelaar ziet is het resultaat na een transformatie: een specifieker model óf code.

1.2.2. Ontwikkelen met MDA: AndroMDA

Zoals in de vorige paragraaf genoemd maken ontwikkelaars, die volgens het MDA framework ontwikkelen, sterk gebruik van tooling. Denk aan een modelleertool voor het opstellen van een PIM model in UML, of het transformeren van een model naar programmacode. In de verschillende fases nemen tools een hoop werk over van de ontwikkelaar.
Een tool die werkt volgens het MDA framework is *AndroMDA* [AndroMDA2006]. Bij het onderzoek naar de toepasbaarheid van MDA concepten speelt deze open-source MDA tool een centrale rol en is gedurende het afstudeerproject gebruikt.
AndroMDA is in het MDA ontwikkeltraject de code generator die een PIM model (een UML model in XMI formaat) inleest en deze direct transformeert naar code (bijv. Java of .Net). Om een PIM model te transformeren maakt AndroMDA gebruik van zogenaamde *cartridges* waarin de verschillende transformatieregels zijn opgeslagen. Het modelleren zelf vindt plaats in een zelfstandige modelleertool die UML in het XMI formaat kan opslaan. Een cartridge is een belangrijk onderdeel van AndroMDA waarin het UML model wordt omgezet in code (de transformatie definities). AndroMDA beschikt momenteel over een tiental cartridges waarmee uit UML (plus uitbreidingen op de standaard notatie) o.a. Spring classes, Hibernate classes, Struts classes en XML schema's te genereren zijn.
Zoals beschreven bestaat het MDA framework uit twee modellen: een PIM en een PSM. Echter, AndroMDA wijkt hiervan af en maakt slechts gebruik van één model en transformeert deze direct naar code. De afwezigheid van het PSM wordt gecompenseerd door platform specifieke informatie toe te voegen in het PIM model, in de vorm van UML profielen (zie hoofdstuk 1.2.5.). Dit verrijkte model wordt in de documentatie een 'marked PIM' genoemd. Een voorbeeld van zo'n verrijkt UML model is te vinden in bijlage A.

1.2.3. Voordelen van MDA

Het ontwikkelen van software volgens het MDA framework, en daarbij het gebruik van MDA tools, zou volgens verschillende bronnen een aantal voordelen opleveren ten opzichte van 'traditionele' ontwikkeling.

- MDA zorgt er voor dat ontwikkelaars meer bezig zijn met het oplossen van problemen uit de business [Kleppe2003]. Ontwikkelaars modelleren de applicatie op PIM niveau zonder aandacht te besteden aan platform specifieke details.

- MDA zorgt voor een verbetering van de productiviteit, doordat een hoop werk wordt overgenomen door MDA tools [Meservy2005]. Denk daarbij aan de generatie van code, die standaard is vastgelegd in de transformatie definities voor de PSM modellen.
- MDA dwingt ontwikkelaars te ontwikkelen volgens een bepaalde architectuur [Mizuta2005]. De code die vanuit het model wordt gegenereerd kan standaard aan bepaalde kwaliteitseisen voldoen die zijn vastgelegd in de transformatie definities.
- En MDA is 'portable' doordat (in theorie) een PIM naar elk willekeurig specifiek platform kan worden getransformeerd [Kleppe2003]. De komst van een nieuwe programmeertaal betekent in zo'n geval dat 'slechts' een nieuwe transformatie definitie ontwikkeld wordt om de bestaande PIM te transformeren naar een PSM (en vervolgens naar code).

1.2.4. UML als formele taal

De *Unified Modeling Language* (UML) is – net als MDA – één van de standaarden opgesteld door de OMG. Om code te genereren uit UML modellen moeten de modellen een éénduidige betekenis hebben. Een éénduidige definitie in UML is mogelijk door de grammatica van de modelleertaal te beschrijven d.m.v. *metamodeling*. De OMG heeft daarvoor een vier lagen architectuur opgesteld (M0 t/m M3), waarbij M3 het hoogste abstractie niveau vormt [OMGMOF, Kleppe2003]. In deze laag is de *Meta Object Facility* (MOF) als standaard gedefinieerd en is de formele taal om UML modellen mee te beschrijven [Kleppe2003]. De MOF bevat een bibliotheek van alle basiselementen waaruit UML elementen zijn samengesteld. Alle modelleertalen, zoals UML en CWM, zijn instanties van de MOF.

Zoals eerder gesteld mogen modellen die voor MDA tools gebruikt worden niet ambigu zijn: de betekenis van een model moet eenduidig zijn zodat de MDA tool exact weet welke programmacode hij moet genereren. Het opstellen van een metamodel van een model specificeert hoe het model er uit moet komen te zien (soort grammatica regels), en zorgt er dus voor dat elk element uit een model een eenduidige betekenis krijgt en daardoor bruikbaar is binnen het MDA concept. Kennis over metamodeling is dus vereist voor in het onderzoek [Kleppe2003].
Daarnaast wordt er getracht in het onderzoek een transformatie definitie op te stellen in de vorm van een AndroMDA cartridge. De regels uit zo'n definitie zijn van toepassing op het metamodel en vereisen ook daarom kennis over de elementen uit het metamodel.

1.2.5. UML extensies

Het modelleren van applicaties of technologieën vereist vaak een domein specifiek model (met eenduidig gedefinieerde semantiek). De standaard UML notatie biedt vaak beperkte mogelijkheden om de domein specifieke informatie uit te drukken. Een voorbeeld is het modelleren van de processen van een workflow management systeem in UML [Schleicher2001].
Om UML te gebruiken voor domein specifieke technologieën, kan UML op twee manieren worden uitgebreid om zodoende een hoger uitdrukkingsvermogen te verkrijgen. In [Lopez2004] kenmerkt men dit als '*heavyweight extension*' en '*lightweight extension*'. Bij de eerst genoemde wordt de bibliotheek van de MOF uitgebreid. Dit zorgt voor een eigen implementatie van UML (bijv. de toevoeging van nieuwe grafische elementen) en is dus geen UML meer (zoals gespecificeerd door de OMG). Deze manier van uitbreiding (om een domein specifieke technologie te kunnen uitdrukken) vereist dus een nieuw type modelleertool, met een eigen modelleertaal. Een voorbeeld van dit type uitbreiding is de MDA tool *OptimalJ*, die beschikt over een eigen modelleeromgeving én niet-standaard grafische elementen.
Bij 'lightweight extension' wordt er een ander mechanisme gebruikt om UML uit te breiden: *UML profielen*. Dit mechanisme brengt geen wijzigingen aan in de standaard MOF bibliotheek en is daarom bruikbaar in elke willekeurige UML tool (die UML profielen ondersteund). AndroMDA beschikt niet over een eigen modelleeromgeving, maar gebruikt een losstaande UML tool. Hierdoor wordt de keuze voor het uitbreiden van UML beperkt[2] tot de lightweight variant, in de vorm van een UML profiel.

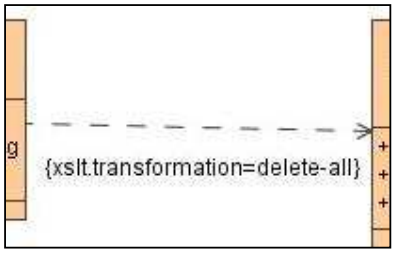
[2] Het kiezen voor de heavyweight variant zou betekenen dat er een eigen modelleer omgeving geprogrammeerd moet worden, wat - gezien de korte periode - niet realistisch is.

Een UML profiel is in feite een UML model dat geïmporteerd wordt in een ander UML model, waarbij de elementen van de import 'read-only' zijn. Deze elementen van een UML profiel zijn van tevoren opgesteld met als doel een uitbreiding van de standaard UML notatie te maken, zodat domein specifieke informatie kan worden gemodelleerd. Een UML profiel bestaat hoofdzakelijk uit de volgende elementen.

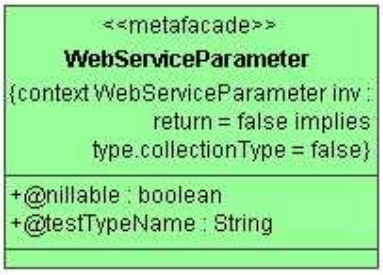
Figuur 2 - *De operaties van de entiteit Order worden als WebService geïmplementeerd.*

Stereotypes
Stereotypes [Berner1999] is één van de meest voorkomende manieren van UML uitbreiding en wordt boven de naam van de klasse genoteerd met 'vishaken' (zie figuur 2). Het is bedoeld om objecten binnen een klassendiagram te classificeren naar gebruik en helpt bij het organiseren van elementen in een model.

Tagged Values
Een tagged value is een andere manier om een model te verrijken. Tagged values worden niet gebruikt voor classificaties - zoals bij stereotypes – maar om informatie toe te voegen in een model. Een tagged value bestaat uit een naam en één of meerdere parameters, genoteerd tussen twee accolades (zie figuur 3). Voor het ingeven van een parameter typt de ontwikkelaar zelf een waarde in of maakt gebruik van een keuzebox. In bijlage A is een tagged value aangebracht in de compositierelatie.



Figuur 3 – *De tagged value geeft extra informatie over deze dependency.*



Figuur 4 - *OCL regels worden direct onder de naam van de entiteit getoond.*

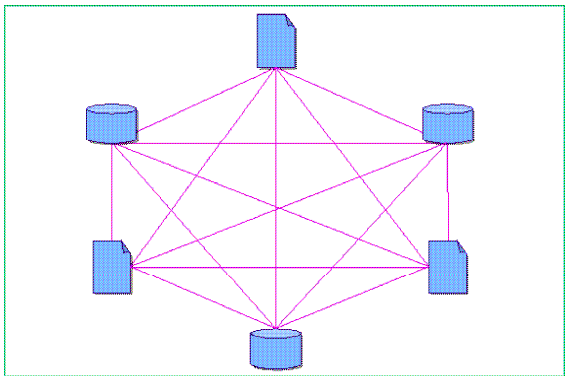
Constraints
Constraints leggen beperkingen op aan het model of zijn geschikt om modellen te valideren op de juiste syntax [Gogolla2003]. Constraints worden in UML modellen aangebracht d.m.v. Object Constraint Language (OCL) regels [Warmer2003] (zie figuur 4). Voorgedefinieerde regels in het UML profiel kunnen erg handig zijn. Model ontwikkelaars die geen kennis hebben van OCL, kunnen dan toch gebruik maken van de voorgedefinieerde regels uit het profiel.

1.3. Probleembeschrijving en onderzoek

Om een oordeel te kunnen geven over de bruikbaarheid van de MDA concepten én de tool AndroMDA op het gebied van datatransformaties (zie onderzoeksvraag), is er een probleem geïdentificeerd binnen Ordina dat opgelost kan worden met AndroMDA. Het probleem betreft de integratie van systemen, waarbij de gegevens tussen de systemen getransformeerd moeten worden om op juiste wijze met elkaar te kunnen communiceren. Een oplossing van het probleem vormt niet alleen een meerwaarde voor Ordina, maar geeft tevens inzicht in de toepassing en bruikbaarheid van AndroMDA. De volgende paragraaf beschrijft dit probleem, wat als case dient voor AndroMDA.
Het volgende hoofdstuk beschrijft de context waarin het probleem zich afspeelt, waarna hoofdstuk 1.3.2. de voorgestelde oplossing beschrijft en hoe deze case kan helpen bij het beantwoorden van de onderzoeksvraag.

1.3.1. Integratie SOA: Service Oriented Integration

Een nieuwe technologie waar bedrijven momenteel veel in investeren is *Service Oriented Architecture* (SOA): een verzameling van tools, methodes, richtlijnen en technieken om IT- en businessservices op elkaar te laten aansluiten (meer details zie bijlage B). Een belangrijk aspect in SOA zijn de services, waarbij legacy systemen, externe webdiensten en nieuwe Java applicaties als services met elkaar communiceren via één centraal punt: de *Enterprise Service Bus* (ESB).
Elk (legacy)systeem communiceert data volgens zijn eigen dataschema via de ESB. Hetgeen data transformaties vereist tussen de systemen. Bij een klein aantal systemen wordt volstaan met een transformatie tussen systeem X naar elk ander systeem: een point-to-point transformatie (zie figuur 5). Deze manier van integreren is goedkoop en noemt men *Web Service Integration* (WSI) [Newcomer2005, p172]. Echter, bij een veelvoud aan systemen die data uitwisselen neemt het aantal data transformaties explosief toe en is een andere integratie techniek gewenst.



Figuur 5 - *Bij WSI neemt de point-to-point transformatie explosief toe bij uitbreiding van een nieuwe node.*

Service Oriented Integration (SOI) speelt bij complexe SOA infrastructuren een belangrijke rol waarbij gebruik wordt gemaakt van een domeinmodel[3]. Data tussen twee systemen worden in de ESB eerst vertaald naar het domeinmodel en vervolgens weer vertaald naar het dataschema voor het doelsysteem. Deze tussenstappen lijken in eerste instantie omslachtig vergeleken met WSI, maar bieden duidelijke voordelen. Ten eerste is er een definitie vanuit de business waar bijvoorbeeld een entiteit “Order” aan moet voldoen. Nieuwe ontwikkelde diensten conformeren zich aan deze standaarden vanuit de business. Ten tweede blijft de groei van (oude) systemen beheersbaar: bij het aansluiten van een legacy systeem aan de ESB hoeft slechts één transformatie naar het domeinmodel te worden gerealiseerd. Een grafische weergave van deze situatie is te vinden in bijlage C.

1.3.2. Datatransformatie d.m.v. AndroMDA

Ordina levert in samenwerking met Cordys SOA oplossingen volgens SOI. De berichten tussen de systemen (en domeinmodel) worden verstuurd in XML. Omdat het formaat van een XML bestand verschilt tussen de systemen, dient een transformatie te zijn aangegeven (zie bijlage C). Een gebruikte techniek hiervoor is XSLT templates [Hunter2003, p87]. Deze transformatie techniek wordt ‘met de hand’ geschreven door ontwikkelaars. Ordina heeft aangegeven beter gebruik te

[3] Ook wel het businessmodel genoemd. Wordt ontworpen vanuit het oogpunt van de bedrijfskundigen.

willen maken van tools en/of technieken voor het automatiseren van transformatie code. Een eis daarbij is dat de tools en/of technieken ondersteuning moeten bieden bij het ontwikkelen (verkorte ontwikkeltijd) én kunnen omgaan met een gewijzigd domeinmodel. Een wijziging in het domeinmodel betekent in alle gevallen dat de XSLT templates voor alle aangesloten systemen worden aangepast.
Voor deze ondersteuning is AndroMDA voorgesteld om XSLT te genereren. In een PIM model wordt het technische model en het domeinmodel gemodelleerd, in de vorm van twee dataschema's. Tussen de twee schema's wordt aangegeven hoe de transformatie plaatsvindt. Vervolgens leest AndroMDA het PIM model in en wordt er XSLT gegenereerd door de twee dataschema's uit het diagram te interpreteren. Een voorbeeld oplossing is in grafische vorm te vinden in bijlage D.
De case van het project is geschikt voor het beantwoorden van de onderzoeksvraag om de volgende redenen. In het PIM model worden twee dataschema's gemodelleerd volgens een eigen ontworpen notatie. Zodoende is het mogelijk een oordeel te geven over het uitdrukkingsvermogen van het PIM model voor dataschema's bij datatransformaties. Daarnaast wordt het PIM model getransformeerd naar XSLT, en kan er een oordeel worden gegeven over de mogelijkheden om transformatiecode te genereren vanuit een PIM met AndroMDA.

1.3.3. Onderzoeksvraag

De onderzoeksvraag van de afstudeeropdracht luidt:

Is het mogelijk de MDA tool 'AndroMDA' te gebruiken om datatransformaties uit te voeren voor Ordina.

Het antwoord op deze vraag wordt verkregen aan de hand van een aantal criteria (zie hoofdstuk 4.2). Deze criteria zijn opgesteld, specifiek voor de problematiek met datatransformaties bij Ordina (zie hoofdstuk 1.3).

1.3.4. Doelstellingen

Het afstudeerproject heeft twee (hoofd)doelstellingen:

Bieden van een oplossing voor het ontwikkelen van transformatie code.
De eerder genoemde integratie methode (SOI) voor SOA oplossingen vereist – ondanks de uitgebreide tooling van Cordys – 'handmatige' ontwikkeling van code om het ene dataschema om te zetten naar een ander dataschema. Deze code is o.a. in XSLT.
Dit onderzoek biedt een manier om deze XSLT code te genereren, met behulp van AndroMDA.

Advies over de bruikbaarheid van AndroMDA.
Het MDA framework en de beschikbare tooling is relatief nieuw en evolueert nog steeds. Ordina J-technologies gebruikt in haar projecten nog geen MDA tooling, maar is wel geïnteresseerd in de mogelijkheden. Om Ordina te adviseren over de mogelijkheden van het gebruik van MDA wordt de tool AndroMDA onderzocht. In dit document wordt hierover een advies gegeven waarbij de zaken 'productiviteit' en 'kwaliteit' een rol spelen, omdat juist deze twee aspecten als waardevol worden beschouwd bij de beslissing of - in de toekomst - AndroMDA in projectontwikkeling wordt gebruikt.

2. Aanpak

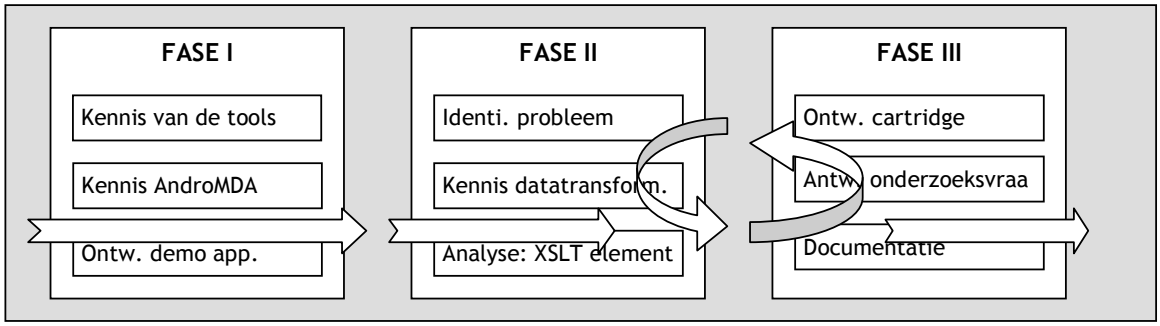
Het afstudeeronderzoek, dat in een periode van drie maanden plaats vindt, is onderverdeeld in drie fases. Elke fase vertegenwoordigt een logische opeenvolgende stap in het gehele onderzoek, waarin een aantal activiteiten is ondergebracht die bij elkaar horen. De fases vormen de leidraad voor de planning en zijn als volgt gedefinieerd:

Fase I – AndroMDA gebruiker
De eerste fase in het onderzoek betreft de kennismaking met de MDA tool AndroMDA. De fase is genoemd als "AndroMDA gebruiker" omdat het leren gebruiken van de tool (en dus het ontwikkelen van applicaties) centraal staat. De grote complexiteit van deze tool en de noodzaak om met AndroMDA goed te kunnen werken vereist het definiëren van een apart ingeplande fase.
Tijdens deze fase wordt er kennis genomen van de installatie en configuratie van AndroMDA én alle andere tools die met AndroMDA samenwerken (build manager voor Java, UML modelleertools, Java webserver). Na de installatie en configuratie wordt er een grote hoeveelheid documentatie bestudeerd. Alhoewel onvolledig, vormt de documentatie over de werking van de cartridges (transformatie definities van model naar code) het belangrijkste deel.
Het doel van deze fase is te leren werken met AndroMDA en vormt de voorbereidende fase voor de rest van het onderzoek. Ook wordt er door de kennis die is vergaard een gedetailleerde planning opgesteld voor de volgende fase: er kan nu een betere schatting worden gemaakt over de haalbaarheid en de richting van de oplossing voor data transformatie.

Fase II – Analyse en oriëntatie probleem data transformatie.
Naast het onderzoek naar de concepten van MDA en de MDA tool AndroMDA, is er voor de start van het project een probleem geïdentificeerd op het gebied van datatransformaties bij service georiënteerde integratie. Daarbij is aangegeven dat MDA een rol zou kunnen spelen om dit probleem op te lossen. Deze fase is genoemd als "Analyse en oriëntatie probleem data transformatie" omdat in deze fase het probleem beter wordt geanalyseerd en verder wordt uitgediept om vervolgens een concept oplossing te ontwikkelen.
De activiteiten in deze fase zijn analytisch van aard en zijn op te delen in twee onderdelen. Als eerste wordt de problematiek betreffende datatransformatie tussen technische- en conceptuele modellen verder uitgediept, waarbij o.a. wordt gekeken hoe de doel technologie (XSLT) eruit komt te zien en welke transformaties er plaatsvinden. Ook worden ontwikkelaars geïnterviewd en de werkdocumenten uit Ordina projecten bestudeerd om zodoende tot een set van transformatie mogelijkheden te komen.
In het tweede deel van deze fase wordt er gewerkt aan een oplossing met behulp van AndroMDA. Er wordt bestudeerd in hoeverre de standaard UML notatie de verschillende transformaties kan beschrijven, en welke eventuele UML extensies hiervoor nodig zijn. Voor de mogelijkheden en richtlijnen van UML extensies wordt literatuur geraadpleegd.
Het doel van deze fase is te komen tot een UML notatie voor het beschrijven van transformaties tussen twee schema's, en de daarbij behorende transformatie definitie naar de doel technologie (XSLT).

Fase III – AndroMDA ontwikkelaar.
In de laatste fase van het afstudeeronderzoek wordt alle kennis uit de vorige fases toegepast en wordt er een nieuwe AndroMDA cartridge ontwikkeld, die voortaan door gebruikers kan worden ingezet om XSLT te genereren. In tegenstelling tot fase I wordt in deze fase dus een uitbreiding geschreven voor AndroMDA. De fase heet daarom "AndroMDA ontwikkelaar".
Activiteiten in deze fase bestaan uit het ontwikkelen van de AndroMDA cartridge en het opleveren van documentatie zodat ontwikkelaars straks gebruik kunnen maken van de nieuwe cartridge in AndroMDA. Aan het einde van het project wordt er een antwoord gegeven op de onderzoeksvraag door de theorie van datatransformaties te vergelijken met de praktijk, en zodoende conclusies te trekken over de verdere mogelijkheden van AndroMDA op datatransformatie gebied.
Het doel van deze fase is een cartridge op te leveren en een antwoord te geven op de onderzoeksvraag. De cartridge is in staat vanuit een PIM model XSLT code te genereren. In het PIM model moet de transformatie tussen twee datamodellen zijn weergegeven, zodat de AndroMDA cartridge daaruit XSLT kan genereren.

De praktijk
Het project is in werkelijkheid op iteratieve wijze verlopen (zie figuur 6). Vanaf de tweede fase zijn er continue uitgewerkte concepten en oplossingen geïmplementeerd in AndroMDA. De resultaten van de 'proof of concepts' zijn vervolgens steeds weer in een volgend ontwerp gebruikt. Een voordeel van deze iteratieve aanpak was dat de complexiteit beter beheersbaar bleef: functionaliteit is in kleine delen tegelijkertijd ontwikkeld, ervaringen en problemen bij de implementatie in fase 3 konden weer worden gebruikt in fase 2.



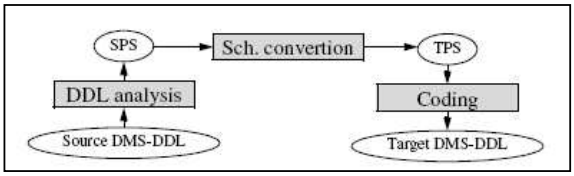
Figuur 6 – *Schematische weergave van projectverloop.*

3. Datatransformaties

Datatransformatie gaat over het transformeren of omzetten van gegevens van het ene informatiesysteem naar het andere informatiesysteem. Daarbij kan het gaan om het eenmalig converteren van informatie uit een legacy systeem naar een nieuw informatiesysteem. Maar ook bij communicatie tussen informatiesystemen speelt datatransformatie een belangrijke rol. Het hele proces van datatransformatie bestaat uit drie fases die zijn ontleend uit [Cleve2005] [4], namelijk schema conversie, data conversie en programma conversie. Dit hoofdstuk bespreekt de drie fases binnen datatransformatie en vormt het theoretische kader over datatransformaties in het algemeen.

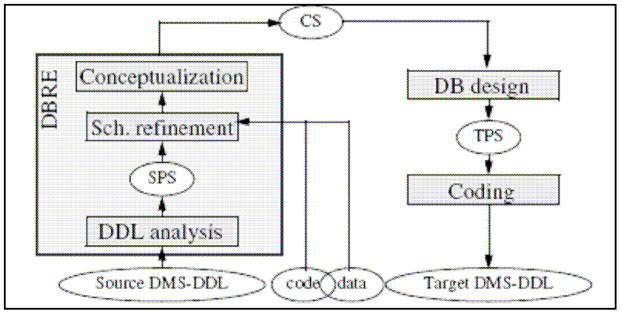
3.1. Schema conversie

Database schema conversie is het proces waarbij een transformatie tool een legacy database schema wordt getransformeerd naar een soortgelijk schema, uitgedrukt in de nieuwe doel technologie. Een schema van een database is een formele beschrijving van de structuur van de database. Het ontwerpen van een schema gaat aan de hand van een ontwerp strategie, bestaande uit verschillende abstractie niveaus [Bommel2005]. Een strategie bestaat uit een conceptueel schema, een logisch schema, een fysiek schema en de code. Een schema kan dus in verschillende abstractie niveaus voorkomen, waarbij het conceptuele schema de werkelijke wereld benadert en het fysieke schema de technische oplossing voor de database beschrijft. Als conceptueel schema is het ook mogelijk een statisch UML diagram te gebruiken [Bommel2005].
De conversie van een schema vindt in twee stappen plaats. Stap één is het opstellen van de schema's en stap twee is het definiëren van de transformatie tussen beiden. Voor het opstellen zijn twee schema's nodig: het bronschema en het doelschema. Deze worden in een transformatie tool grafisch weergegeven om de verschillen en overeenkomsten inzichtelijker te maken en om in de tweede stap de transformaties tussen de schema's aan te geven. Het bron- en doelschema worden o.a. verkregen door het bronschema te importeren uit een bron (database of XML bestand). Dit *fysieke* schema wordt via de ontwerp strategie uit [Bommel2005] getransformeerd naar een conceptueel schema, waaruit het *fysieke* doelschema ontstaat. Of alle abstractie niveaus nodig zijn hangt af van een gekozen conversie aanpak, zoals gedefinieerd in [Henrard2002]. Een populaire aanpak is de 'physical approach' waarbij de bron en doelschema's één-op-één (op fysiek niveau) geconverteerd zijn (zie figuur 7, ontleend aan [Henrard2002]). Zo'n aanpak is makkelijk en kan vrijwel volledig automatisch worden uitgevoerd waardoor de operatie goedkoop is. Echter, alle nadelen van de legacy database zijn terug te vinden in het nieuwe platform: obscure naamgeving en inefficiënte datastructuren uit het oude schema worden overgenomen in het nieuwe schema, waardoor het nieuwe platform niet optimaal kan presteren en moeilijk uitbreidbaar is in de toekomst.



Figuur 7 – *'physical approach'. Hierbij wordt het Source Physical Schema (SPS) direct omgezet naar een Target Physical Schema (TPS).*

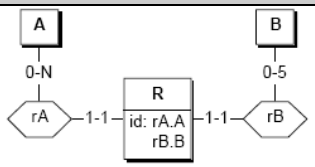
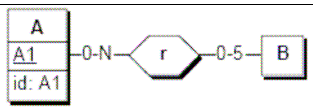
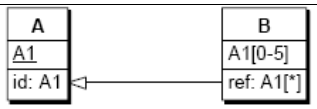
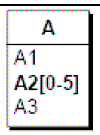
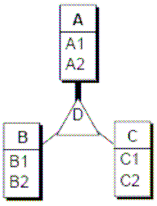
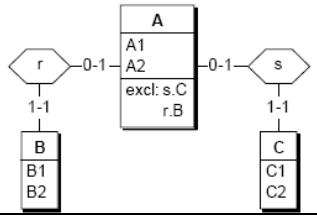
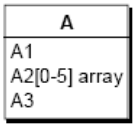
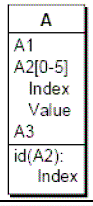
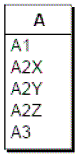
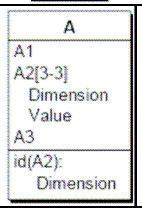
De andere aanpak is de 'semantic approach' (zie figuur 8, ontleend aan [Henrard2002]). Deze aanpak heeft niet



Figuur 8 – *'semantic approach'. Hierbij wordt het Source Physical Schema (SPS) eerst omgezet naar een Conceptual Schema (CS) en daarna naar een Target Physical Schema (TPS).*

[4] Bron [Mork2006] behandeld het proces in vijf fases. Dit komt in feite op hetzelfde neer, omdat in dit document nog een onderverdeling wordt gemaakt in een aantal fases.

dezelfde nadelen als de physical approach omdat het als doel heeft een semantisch identiek schema op te leveren die geoptimaliseerd is voor de nieuwe database. Echter, deze aanpak is wel complexer. Als eerste wordt er – d.m.v. DB Reverse Engineering (DBRE) – het fysieke schema van de bron database (SPS) geëxtraheerd en omgezet naar een conceptueel schema. Daarna wordt het conceptuele schema weer getransformeerd naar de datastructuur van de doel database (TPS). Voor de transformatie van de structuur in een conceptueel schema zijn in [Bommel2005] een zestal transformaties geïdentificeerd, die zijn aangeduid als de *primitieve* transformaties. Alle andere denkbare transformaties kunnen worden samengesteld uit deze zes types. Deze primitieve transformaties zijn in tabel 1 weergegeven. De afbeeldingen zijn ontleend aan [Bommel2005].

Bronschema	Doelschema	Korte beschrijving
		Transformatie van relatie r naar entiteit R.
		Transformatie van relatie r naar een referentie attribuut B.A1 in entiteit B
		Transformatie van attribuut A2 (collectie, set of bag) naar een entiteit EA2.
		Transformatie van een ‘is een’ relatie D naar een één-op-één relatie.
		Transformatie van een array attribuut A2 naar een ‘set-multivalue’ attribuut A2.
		Transformatie van seriële attributen naar een multivalue attribute.

Tabel 1 – *Transformatie van structuur in conceptueel schema.*

De tweede stap in de conversie van een schema is het aangeven van de transformatie van de gegevens. Het aangeven van deze transformatie gaat met behulp van de transformatie tools, en wordt in grafische vorm uitgedrukt. Dat betekent dat een transformatie tool, naast alle elementen van een conceptueel schema (zie tabel 1), alle mogelijke datatransformaties moet ondersteunen. Een probleem hierbij is dat er zich veel verschillende situaties en/of problemen kunnen voordoen bij het omzetten van de gegevens van het ene schema naar het andere schema. Tools als [Mapforce] ondersteunen o.a. logische expressies, wiskundige expressies, string functies en date/time functies (zie tabel 2). Daarnaast blijkt uit eigen ervaring dat de schema’s voor conversie vaak veel velden bevatten, wat bij een semantische aanpak een tijdrovende (en dus een kostbare) klus betekent. Een

tool als Harmony [Mork2006] helpt ontwikkelaars, door met digitale woordenboeken een groot aantal velden uit bron- en doelschema met elkaar te matchen.

Type transformatie	Beschrijving
String transformaties	
• Concatenatie van strings	*De strings van twee of meer velden uit het bronmodel zijn in één veld samengevoegd in het doelmodel.*
• Gedeelte van string	*Van een string uit het bronmodel wordt slechts een bepaald gedeelte (substring) overgezet in het doelmodel.*
• Vervanging van string	*Van een string uit het bronmodel wordt een bepaald gedeelte vervangen (replace) en dan overgezet in het doelmodel.*
Datum / Tijd transformaties	
• Tijdzone transformaties	*De tijd of datum uit het bronmodel wordt aangepast aan de gewenste tijdzone (t.o.v. de huidige tijdzone) en overgezet in het doelmodel.*
• Dag / Maand / Jaar / Uur etc extractie.	*Van de tijd of datum uit het bronmodel wordt alleen de dag / maand / jaar / uur / etc geëxtraheerd en overgezet in het doelmodel.*
• Datum / tijd notatie	*De tijd of datum uit het bronmodel wordt - volgens de aangegeven notatie - overgezet in het doelmodel.*
Wiskundige transformaties	
• Optellen, aftrekken, vermenig-vuldigen, delen.	*Twee of meer numerieke waardes uit het bronmodel zijn opgeteld, afgetrokken, etc en overgezet in één veld in het doelmodel.*
• Afronden	*Een numerieke waarde uit het bronmodel wordt op bepaalde wijze afgerond en overgezet in het doelmodel.*
Condities bij transformaties	
• Bepaalde waarde van een element.	*De transformatie of overzetting van een veld vindt alleen plaats wanneer het veld in het bronmodel een bepaalde waarde heeft.*
• Binnen domein of groep.	*Transformatie vindt plaats wanneer een element uit het bronmodel valt binnen bepaalde grenswaardes. Per domein of groep van waardes is een resultaatwaarde vastgesteld die wordt overgezet in het doelmodel.*
• String matching	*Een string wordt pas overgezt van bron- naar doelmodel indien de string uit het doelmodel overeenkomt (matched) met een bepaalde waarde (dit kan een regular expressie zijn).*
• Vergelijking van twee elementen (logische conditie)	*Transformatie of overzetting vindt plaats wanneer een vergelijking (groter dan, gelijk aan, klein dan, etc) tussen twee elementen uit het bronmodel positief uitvalt.*

Tabel 2 – *Type transformaties tussen bron- en doelmodel.*

3.2. Data conversie

Bij data conversie wordt alle inhoud van de bron database overgezet naar de nieuwe database. Deze conversie bestaat uit het Extract-Transform-Load (ETL) proces, waarbij gebruik wordt gemaakt van de twee schema's (en de mapping daar tussen) uit de vorige conversie fase [Henrard2002]. Eerst worden de gegevens uit de oude database gehaald (extract). Als tweede worden de gegevens op die wijze geformatteerd zoals aangegeven in de mapping (transform). En als laatste worden deze gegevens in de doel database geschreven (load).
Een probleem dat zich kan voordoen bij de *semantic approach* is dat gegevens uit de bron niet voldoen aan de constaints in het nieuwe schema, die tijdens de conceptualisatie zijn gevonden. De corrupte gegevens vereisen dan eerst een correctie voordat de migratie plaatsvindt. Een oplossing kan zijn dat de ontwikkelaar kan ingrijpen tijdens het proces óf dat de tool zelf "correctie strategieën" heeft om de gegevens te corrigeren.

3.3. Programma conversie

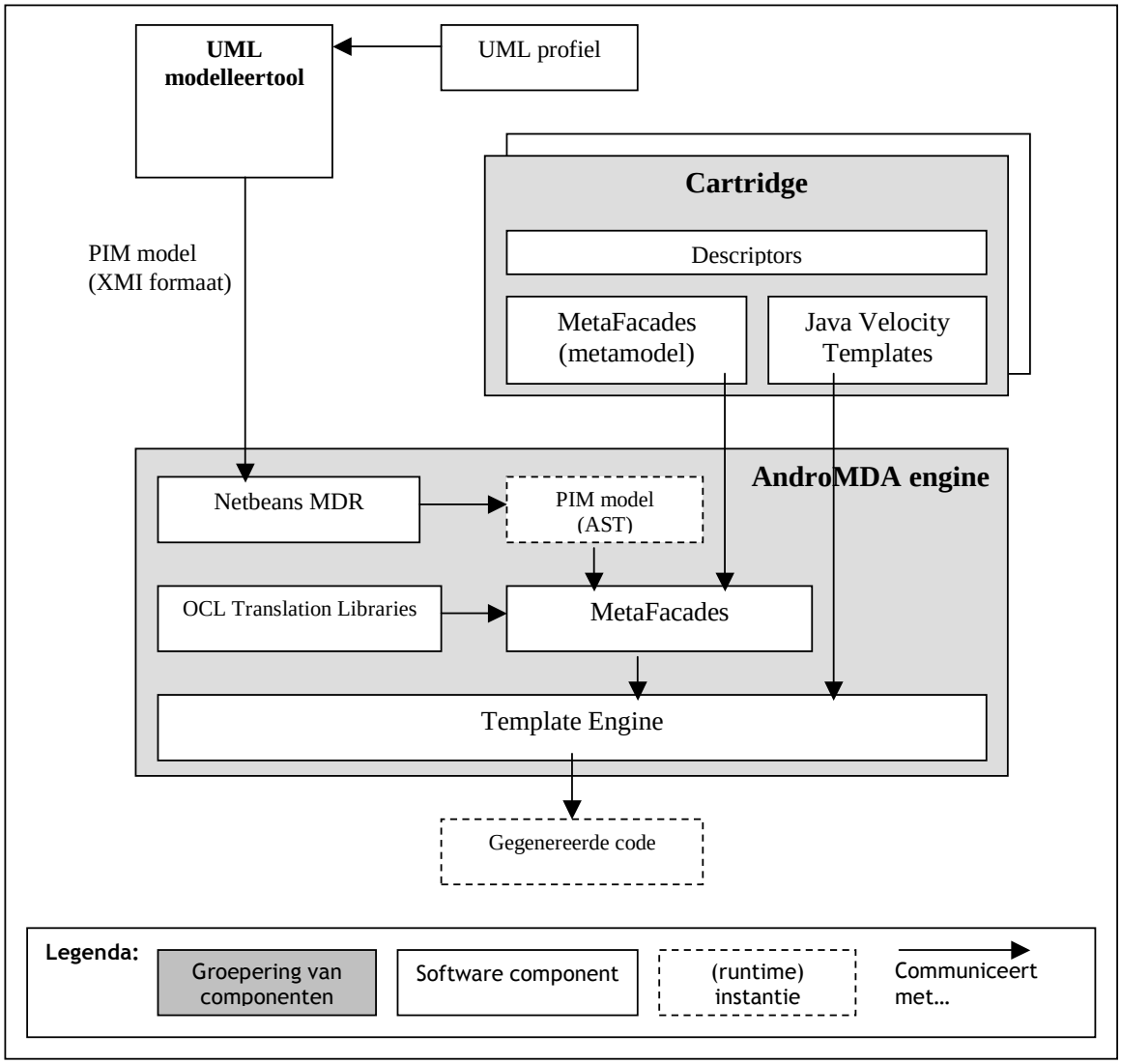
Programma conversie is het wijzigen van de oude software, zodat deze gebruik kan maken van de nieuwe database en gegevens opslaat volgens het nieuwe schema. Het wijzigen van de software is een programma transformatie, en lijkt in eerste instantie buiten de scope van het onderzoek te vallen: Programma transformaties vereisen andere tools dan bij datatransformatie en maken gebruik van herschrijfregels, in plaats van een grafische weergave [Cleve2005] zoals bij datatransformaties.
Echter, de twee type transformaties zijn volgens [Mork2006] en [Cleve2005] nauw met elkaar verbonden, omdat programma conversie nu eenmaal een groot probleem is bij datatransformaties. Wanneer de nieuwe database in gebruik wordt genomen, moet de oude software daar wel gebruik van kunnen maken. Om de programma conversie correct uit te voeren moeten de wijzigingen aan de software consistent zijn aan de wijzigingen die in het dataschema zijn aangebracht. [Cleve2005] identificeert dit probleem en beschrijft dat zowel de schema/data conversie als de programmaconversie parallel moeten verlopen door een mapping tussen beiden aan te brengen.
Naast deze mapping kan de programma conversie op drie manieren plaatsvinden [Henrard2002]. De eerste manier is het gebruik maken van *wrappers*. De nieuwe database wordt geëncapsuleerd door software die de oude database simuleert. De legacy software kan nog steeds gebruik maken van de oude (fictieve) database. De tweede manier is *statement rewriting*. Statements binnen de legacy software zijn dan vervangen door nieuwe statements die met de nieuwe database kunnen communiceren. De laatste manier is *logic rewriting*, waarbij grote delen van de legacy software zijn herschreven om gebruik te kunnen maken van alle features van de nieuwe database.

4. Uitvoering

Dit hoofdstuk beschrijft hoe het onderzoek is uitgevoerd. Daarbij wordt in grote lijnen de indeling van de drie fases aangehouden.

4.1. Ontwikkelen met AndroMDA

De MDA tool AndroMDA [AndroMDA2006] is – zoals eerder genoemd – een open-source Java tool en wordt door een select team van ontwikkelaars onderhouden. De documentatie bestaat voornamelijk uit een veelvoud aan webpagina's. Daarnaast is het AndroMDA forum [AndroMDAforum] één van de belangrijkste informatiebronnen voor gebruikers.
AndroMDA biedt zelf geen functionaliteit voor het modelleren van UML ('marked PIM') modellen, maar leest een geëxporteerd UML model in, dat in het XMI formaat (versie 1.1 of 1.2) wordt aangeboden. AndroMDA genereert vervolgens de code aan de hand van het UML model en de configuratie van de verschillende componenten binnen AndroMDA. In figuur 9 is de architectuur, en daarin alle componenten, van AndroMDA weergegeven. Een gedegen kennis van deze architectuur is vereist om uitbreidingen te kunnen ontwikkelen voor AndroMDA.



Figuur 9 – *AndroMDA architectuur*

AndroMDA is gebaseerd op de *Netbeans Metadata Repository* (MDR) [NetbeansMDR]. MDR is een Java implementatie van OMG's MOF en laadt alle elementen uit het UML model, als Java objecten, in het geheugen. Deze instantie van het model in het geheugen heet de *Abstract Syntax Tree* (AST). Elke node uit de AST is een Java object die een bepaalde interface implementeert uit de MDR (bijv. UMLClass, Attribute, Association, etc).
Naast het UML model instantieert AndroMDA alle cartridges. De cartridges bevatten – volgens de concepten van MDA – de transformatie definities in de vorm van Java Velocity templates (zie hoofdstuk 4.2.4), en vertellen de template engine hoe het UML model wordt omgezet in code. Cartridges zijn in figuur 9 buiten de AndroMDA engine weergegeven omdat cartridges (als .jar bestand) als plugin te gebruiken zijn binnen AndroMDA. Dit mechanisme stelt ontwikkelaars in staat zelf cartridges te ontwikkelen. Meer details over de inhoud én het ontwikkelen van een cartridge is te vinden in hoofdstuk 4.3.
AndroMDA loopt vervolgens langs alle AST objecten ('AST traversing') en biedt de Velocity templates toegang tot de UML objecten om daar informatie uit te halen voor het code generatie proces. Echter, de objecten bieden een enorme hoeveelheid informatie die de toegang tot die objecten nodeloos complex maakt. Het gebruik van facades schermt deze complexiteit af en zit als 'eenvoudige interface' tussen de AST en de template. Men noemt deze interfaces 'metamodel facades', verkort '*metafacades*' [AndroMDAmetafacades].
De uiteindelijke gegenereerde code is ingedeeld in twee categorieën, afhankelijk van de template configuratie in de descriptor files. De ene categorie bevat gegenereerde code die bij een nieuwe transformatie ronde wordt vervangen. De andere categorie bestaat uit éénmalig gegenereerde code die daarna niet meer opnieuw wordt gegenereerd. Deze code bevat stubs die door de ontwikkelaar zelf geïmplementeerd dient te worden.

Features
AndroMDA kan tijdens het transformatie proces gebruik maken van *translation libraries*. Translation libraries worden gebruikt om *Object Constraint Language* (OCL) [Warmer2003] expressies te vertalen naar een andere programmeertaal. OCL expressies zijn bijvoorbeeld inzetbaar bij het ontwikkelen van metafacades, om regels op te stellen waar een model aan moet voldoen. De OCL expressies worden dan vertaald naar Java code. Meer informatie hierover in hoofdstuk 4.2.

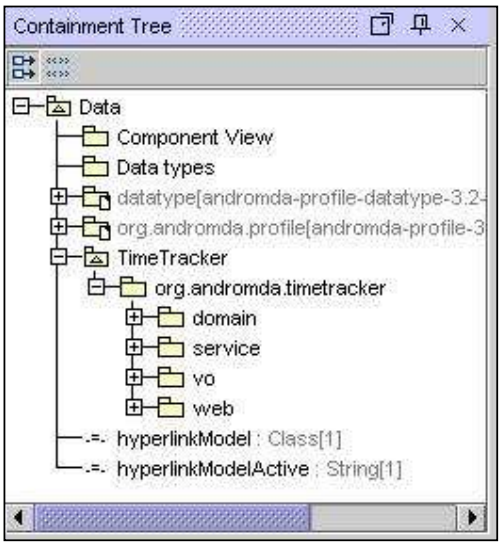
Bij AndroMDA doet zich een trend voor die men ook bij andere open-source producten ziet. AndroMDA maakt sterk gebruik van andere open-source producten om de gehele functionaliteit te bieden. Basiskennis van alle omringende producten is vereist om gebruik te kunnen maken van AndroMDA en maakt het gebruik ervan complex. De volgende paragraaf benoemt de belangrijkste tools en geeft aan welke zaken hierin complex zijn.

4.1.1. Ontwikkel tools

Maven 1.x
[Maven1.x] is een build management tool voor Java projecten (opvolger van Ant) en zorgt voor de aansturing van AndroMDA: via de commandline worden de maven commando's uitgevoerd om AndroMDA taken te laten uitvoeren. Elk ontwikkelproject met AndroMDA is in feite een Maven project.
Bij een transformatie van het UML model start Maven AndroMDA, compileert de gegenereerde code en bouwt vervolgens een enterprise archive (.ear) file. Ook het installeren van AndroMDA vereist alleen Maven: Het instellen van de AndroMDA repository in de `build.properties` file download de laatste versie van de tool en alle benodigde plugins.
Afgezien van de commando's om projecten te kunnen bouwen, is basiskennis van Maven scripts vereist. Een zelf gemaakt Maven script kan procedures in het testproces automatiseren, wat het ontwikkelen van de software makkelijker maakt.

MagicDraw 9.5

[MagicDraw] is een commerciële UML modelleertool van de fabrikant No Magic en wordt gebruikt om alle PIM modellen (in UML) voor AndroMDA te modelleren[5]. Op het moment van schrijven worden alleen class diagrammen en activity diagrammen gebruikt voor AndroMDA. MagicDraw ondersteunt meer soorten diagrammen.



Figuur 10 – *Navigatieboom waarin alle elementen van het project staan geordend.*

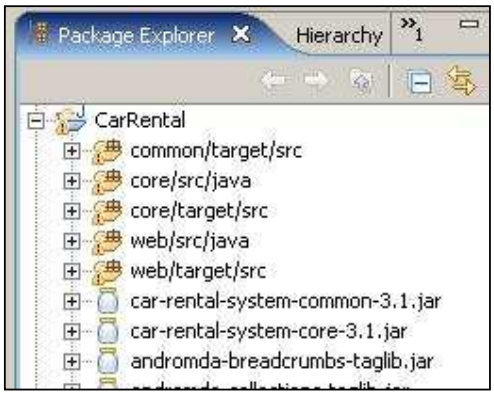
Alhoewel het mogelijk is een alternatieve UML tool te gebruiken (zoals ArgoUML) moet deze over een aantal functies beschikken om te gebruiken voor AndroMDA. MagicDraw beschikt over deze functies: De tool biedt ondersteuning voor UML profielen en kan de modellen opslaan in het XMI formaat. Een andere belangrijke feature is de ondersteuning van de OCL om regels op te leggen aan elementen in een klasse diagram.

Het opstellen van een klasse diagram lijkt gemakkelijk, maar het opstellen van UML elementen én hun relaties vereisen grote precisie van de ontwikkelaar. Daarbij moet de package structuur in het UML model voldoen aan de notatie eisen van elke AndroMDA cartridge (zie figuur 10).

Eclipse 3.1 IDE

Eclipse is een populaire ontwikkelomgeving voor Java en wordt gebruikt om de gegenereerde code uit AndroMDA te wijzigen. Een Maven project is gemakkelijk in Eclipse in te laden, waarna Eclipse de reeds aangemaakte mappen structuur van Maven toont. Gegenereerde code is niet zomaar aanpasbaar. Handmatige wijzigingen worden weer ongedaan gemaakt wanneer AndroMDA opnieuw de code genereert uit het PIM model. Om dit te voorkomen maakt AndroMDA onderscheid tussen volledig gegenereerde code (*target files*) en eenmalig gegenereerde code (*source files*), beiden aangegeven in figuur 11. Deze structuur is tevens zichtbaar in Eclipse, zodat ontwikkelaars makkelijk onderscheid kunnen maken tussen wat wél en wat niet handmatig mag wijzigen (zie figuur 11).

De code die AndroMDA genereert moet niet worden beschouwd als programma code die 'af' is: ontwikkelaars hebben vaak nog de IDE nodig om bepaalde *stubs* te implementeren om verbindingsbruggen te leggen tussen verschillende architectuur lagen (bijv. tussen de business laag en de presentatie laag). AndroMDA moet daarom vooral worden gezien als code generator bij het ontwikkelen van applicaties, in plaats van een volledige applicatie generator: de tool kan veel werk uit handen nemen door alle standaard code te genereren, maar programmacode om niet-standaard functionaliteit te implementeren wordt aan de programmeur overgelaten.



Figuur 11 – *Project structuur van source (src) en target code. Alleen source code is aanpasbaar.*

[5] Tijdens het onderzoek is gebruik gemaakt van de *Community Edition*: een gratis variant met een aantal beperkingen.

4.2. Datatransformatie: analyse en oplossing

In hoofdstuk 4.1 is kennis gemaakt met AndroMDA en de bijbehorende tools. Dit hoofdstuk beschrijft in detail de tweede en derde fase uit het project, zoals beschreven in hoofdstuk 2 (aanpak). In de eerste gedeelte geeft een algemene beschrijving over datatransformaties, waarna een analyse van het probleem wordt gemaakt. Het tweede gedeelte (vanaf hoofdstuk 4.2.4.) geeft een uitwerking van de oplossing.

4.2.1. Scope

In dit onderzoeksproject wordt er een cartridge (als uitbreiding) voor AndroMDA gemaakt die in staat is vanuit een PIM model XSLT (versie 1.0) te genereren. De cartridge bevat dus de transformatie definitie om een XSLT bestand uit een PIM te genereren. Het XSLT bestand dat wordt gecreëerd is zelf ook een transformatie definitie [Hunter2003]. Het is in staat delen van een XML bestand (de input) te transformeren in bijvoorbeeld een XML bestand met een ander formaat. Beide kunnen worden beschouwd als varianten van transformaties. Bij variant één is AndroMDA *direct* als transformatie tool gebruikt. Bij de tweede variant *genereert* AndroMDA transformatie code (in deze case XSLT) dat weer door een aparte XSLT parser wordt ingelezen. In het laatste geval is de MDA tool beter te beschouwen als *hulp* in het transformatieproces.

In het onderzoek wordt er gekeken naar de tweede variant van transformeren: AndroMDA wordt niet direct als transformatie tool ingezet, maar als hulpmiddel. Er is voor deze variant gekozen omdat de uitgevoerde case van deze variant is, en zodoende kan er een antwoord worden gegeven op deze variant van de onderzoeksvraag. Een andere reden is dat AndroMDA niet is ontworpen als transformatie tool, maar als code generator. AndroMDA is bijvoorbeeld niet in staat – in tegenstelling tot echte transformatie tools – het Extract-Transform-Load (ETL) proces uit te voeren en kent als input slechts één UML model in het XMI formaat.

4.2.2. Evaluatie criteria

Om een goede oplossing te ontwikkelen dienen de criteria in kaart te zijn gebracht. Aan de hand van de opgestelde criteria kan de onderzoeksvraag worden beantwoord: Een implementatie van alle criteria resulteert namelijk in een positief antwoord op de onderzoeksvraag.
De evaluatie criteria zijn onderverdeeld in **transformatie criteria** en **notatie criteria**. De één zegt iets over de functionaliteit van de oplossing, de ander iets over hoe de oplossing in UML eruit komt te zien (het PIM model). De twee groepen criteria zijn gescheiden omdat beiden apart van elkaar te implementeren zijn. Ook wordt er op deze wijze expliciet aandacht gegeven aan elke groep, zonder de concepten met elkaar te verwarren.

Transformatie criteria

- *Transformatie mogelijkheden van XML bestanden [prioriteit: hoog]*

Om XSLT templates te genereren zijn de bron- en doelschema's nodig. Tussen het bron- en het doelschema zijn de volgende transformaties gedefinieerd en in tabel 3 weergegeven. De criteria zijn gevonden na een zorgvuldige analyse van één specifiek project van een klant bij Ordina. Tevens is per transformatie criterium vermeld of deze 'standaard' is of 'uitzonderlijk'. Dit geeft globaal aan hoe vaak de betreffende transformatie voorkomt, waarbij de categorie 'standaard operatie' vaak voorkomt. Deze categorie zal dus zeker geïmplementeerd moeten worden om een bruikbare oplossing te krijgen. Van de categorie 'uitzondering' zal waarschijnlijk slechts één criterium worden geïmplementeerd: deze transformaties komen weinig voor en zijn bovendien erg complex om uit te voeren (vanuit dit project, met het gebruik van een PIM model).
Voor de criteria uit de tabel is slechts gebruik gemaakt van een aantal voorbeeld entiteiten ('customer', 'order', etc) ter illustratie.

Beschrijving transformatie	Bron XML data	Doel XML data
Concatenatie van waarde uit één of meerdere elementen naar één element. Namen van de elementen kunnen verschillen. Mogelijkheid tot invoegen van extra strings tussen de waardes. **Categorie: *standaard operatie***	<customer> <first>piet</first> <middle>de</middle> <last>Vries</last> <month>12</month> <year>2006</year> </customer>	<klant> <firstName>piet</firstName> <lastName>de Vries</lastName> <period>12-2006</period> </klant>
Transformeren van gelaagde structuur naar een platte structuur. Namen van de elementen kunnen verschillen. **Categorie: *standaard operatie***	<customer> <first>piet</first> <middle>de</middle> <last>Vries</last> <order> <datum>1-1-06</datum> <status>verzonden</status> </order> </customer>	<customer> <first>piet</first> <middle>de</middle> <last>Vries</last> <date>1-1-06</date> <status>verzonden</status> </customer>
Transformeren van waarde uit bronmodel naar een ander formaat in het doelmodel: gebruikelijk bij datumnotatie. **Categorie: *uitzonderlijk***	<customer> <first>piet</first> <timestamp>120620061836</ti..> </customer>	<customer> <first>piet</first> <date>06122006</date> </customer>
Transformeren van de namen van de elementen, waarbij de gelaagde structuur behouden blijft. **Categorie: *standaard operatie***	<customer> <first>piet</first> <middle>de</middle> <last>Vries</last> <order> <date>1-1-06</date> <status>verzonden</status> </order> </customer>	<klant> <first>piet</first> <middle>de</middle> <last>Vries</last> <bestelling> <datum>1-1-06</datum> <status>verzonden</status> </bestelling> </klant>
Transformeren van elementen die oorspronkelijk niet in het bronmodel voorkomen, maar wel in het doelmodel. **Categorie: *standaard operatie***	<customer> <first>piet</first> <middle>de</middle> <last>Vries</last> </customer>	<customer> <first>piet</first> <middle>de</middle> <last>Vries</last> <status /> </customer>
Creatie van het element in doelmodel aan de hand van waarde uit het bronmodel. **Categorie: *uitzonderlijk***	<customer> <first>piet</first> <clienttype>2</clienttype> <company>Ordina</company> <status>active</status> </customer>	<customer> <first>piet</first> <ITcompany> <name>Ordina</name> <status>active</status> </ITcompany> </customer>
Transformeren van modellen met gelaagde structuur, waarbij een één-op-veel relatie aanwezig is. *Elementen van*	<customer> <name>Bart</name> <orders> <order>	<klant> <name>Bart</name> <bestellingen> <order>

bron- naar doelmodel blijven op hetzelfde niveau. **Categorie: *standaard operatie***	<orderID>23</orderID> <orderDate>0606</ord> </order> <order> <orderID>24</orderID> <orderDate>0706</ord> </order> </orders> </customer>	<nummer>23</ nummer> <orderDate>0606</ord> </order> <order> <nummer>24</ nummer> <orderDate>0706</ord> </order> </bestellingen> </klant>
Transformeren van modellen waarbij waardes uit het bronmodel d.m.v. een *conditie* worden getransformeerd naar een waarde in het doelmodel. **Categorie: *uitzonderlijk***	<customer> <name>Bart</name> <order> <orderID>23</orderID> <orderDate>0606</ord> <status>1</status> </order> </customer>	<customer> <name>Bart</name> <order> <orderID>23</orderID> <orderDate>0606</ord> <status>accepted</status> </order> </customer>

Tabel 3 – *Transformatie criteria: type transformaties voor XML bestanden.*

- *Valide XSLT templates [prioriteit: hoog]*

De gegenereerde XSLT templates zijn valide wanneer deze voldoen aan de syntax regels, opgesteld door de W3C. Valide XSLT templates zijn bruikbaar binnen de Cordys tooling.

Notatie criteria

- *Notatie van PIM model moet intuïtief zijn [prioriteit: middel]*

In het PIM model zijn twee XML schema's gemodelleerd en is de transformatie tussen beiden aangegeven. Hieruit kan AndroMDA XSLT genereren. Vanuit de gebruikerskant is aangegeven dat in het PIM model het uitdrukken van de transformatie en schema's gemakkelijk en intuïtief moet zijn. Voor een intuïtieve notatie wordt gekeken naar reeds bestaande PIM modellen in AndroMDA. Tevens worden er richtlijnen uit de literatuur gebruikt voor het classificeren van schema's. De beoordeling van dit criterium gebeurt aan de hand van gesprekken met ontwikkelaars.

- *Verschil tussen type XML schema's (bron en doelmodel) is zichtbaar in notatie [prioriteit: hoog]*

Beide XML schema's (bron- en doelschema) worden voor de XSLT generatie in de PIM gemodelleerd. Een belangrijk criterium is dat er verschil tussen de twee type schema's in de UML notatie moet worden aangebracht, zodat in één oogopslag duidelijk is welke het bronschema is, én welke het doelschema.

- *Vrije keuze in UML tooling [prioriteit: hoog]*

De notatie die wordt bedacht voor het PIM model mag alleen standaard UML elementen bevatten (volgens het MOF metamodel, hoofdstuk 1.2.4), omdat het model compatible moet zijn met alle UML tools. Ontwikkelaars hebben hierdoor de keuze hun favoriete UML tool te gebruiken. Om toch alle domein specifieke elementen weer te geven in het PIM model wordt er gebruik gemaakt van UML profielen [Schleicher2001].

- *Notatiewijze wordt gecontroleerd met OCL [prioriteit: middel]*

De *Object Constraint Language* wordt vaak gebruikt in de wetenschappelijke literatuur [Gogolla2003, Schleicher2001, Akehurst2002] om de UML notatie te valideren. In dit onderzoek kan OCL een rol spelen om het PIM model te controleren op een juiste notatiewijze. Ontwikkelaars worden op die manier ondersteund bij het modelleren van de PIM en krijgen debug informatie wanneer het PIM inconsistent is. Voor de UML notatie wordt het volgende gevalideerd:

  - Gebruik van juiste set stereotypes.
  - Aanbrengen van associaties tussen class elementen gebeurt op dezelfde wijze zoals reeds bestaande XML schema cartridge in AndroMDA.

- Modelleren van de transformatie in het PIM model kan alleen tussen bronschema en doelschema.

4.2.3. Ontwikkelen van een oplossing

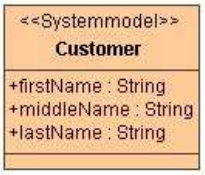
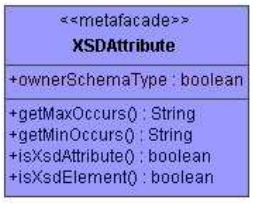
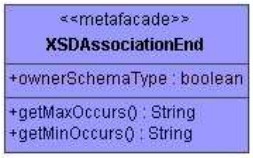
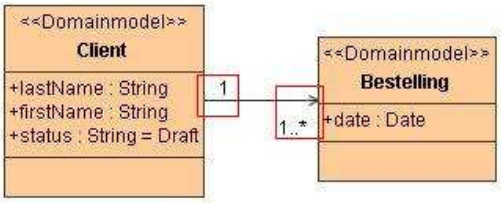
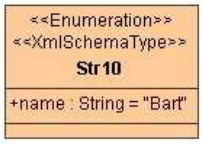
De oplossing is complex. De reden voor de complexiteit is het feit dat de oplossing geen standaard AndroMDA cartridge betreft. In een standaard cartridge wordt van elke UML element uit het PIM een Java bestand gegenereerd, die een standaard code blok bevat (van een specifieke Java technologie). Deze manier van genereren is 'één-op-één'. De cartridge, als resultaat uit dit onderzoek, genereert van alle elementen uit het PIM slechts één bestand: een XSLT template. Daarvoor worden allerlei transformatieregels geïmplementeerd om het PIM model op juiste wijze te interpreteren.

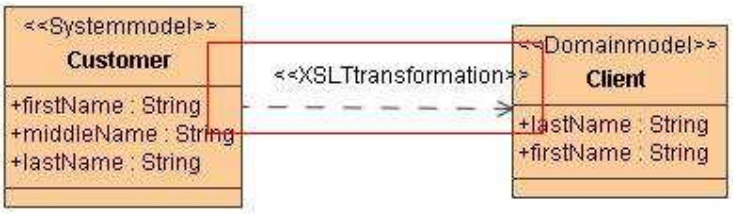
De AndroMDA cartridge, om vanuit het PIM model XSLT te genereren, wordt in een aantal stappen beschreven. De volgende paragrafen bevatten een uitwerking van deze stappen. De eerste stap beschrijft het metamodel van het PIM model. Dit is de eerste stap omdat het metamodel grotendeels wordt overgenomen uit de bestaande AndroMDA cartridge. De bestaande cartridge biedt reeds een intuïtieve notatie voor het PIM model en een metamodel voor de XML schema's.
Als tweede stap worden alle fundamentele XSLT elementen geïdentificeerd. Uit de bron [Hunter2003] is een minimale set van elementen geselecteerd die in staat is de transformatie criteria uit te voeren.
De derde stap is complex. Hier wordt de brug geslagen tussen de twee resultaten uit de vorige stappen: de transformatie definities. De mapping tussen het PIM en XSLT is nodig zodat AndroMDA weet welke UML element uit het PIM naar welk XSLT element gegenereerd wordt.
De overige stappen beschrijven de OCL regels en het UML profiel voor het PIM model.

Metamodel
De AndroMDA metafacade bibliotheek is een voorgedefinieerde Java API, en wordt gebruikt om toegang te krijgen tot de elementen uit het PIM model. Zo is er voor elk UML element (klasse, associatie, attribuut, etc) een metafacade om de specifieke element gegevens op te vragen [AndroMetaAPI]. De metafacades verbergen daarbij de extra complexiteit: ze voorzien ontwikkelaars van een eenvoudige interface tot het model. De metafacades zijn vervolgens - als Java objecten - uit te lezen in de Velocity templates. De metafacades zijn in het architectuur schema (figuur 9) terug te vinden binnen de AndroMDA engine. Een volledige documentatie over metafacades is te vinden in [AndroMDAmetafacades].
Naast de standaard API is het mogelijk *zelf* metafacades te schrijven voor het PIM model: een eigen metamodel. Een reden om deze zelf te ontwikkelen is het wijzigen of uitbreiden van het gedrag van een element: het eigen metamodel is dan een *specialisatie* van de bestaande API. Dit metamodel is in het architectuur schema (figuur 9) terug te vinden in de cartridge.

In dit project is er voor gekozen om een eigen metamodel op te stellen. De reden hiervoor is dat de gemodelleerde XML schema's in het PIM model extra 'gedrag' hebben dat niet in de standaard metafacades API is opgenomen. Bij het opstellen van de metafacades voor het metamodel vertaald elk domein specifiek object zich naar een metafacade. Deze metafacade (in feite een Java klasse) implementeert dan specifieke functionaliteit. Tabel 4 toont de metafacades. Het gehele metamodel (als specialisatie op de standaard AndroMDA API) is te vinden in bijlage E.

Merk tevens op dat er in de onderstaande tabel gebruik is gemaakt van het classificeren van PIM elementen d.m.v. stereotypes (informatie over stereotypes zie hoofdstuk 1.2.5). Realisatie van deze stereotypes (en de rest van het UML profiel) wordt verderop in dit hoofdstuk besproken.

Metafacade	Beschrijving + afbeelding element in PIM
	Vertegenwoordigt een <xsd:complexType> element uit een XML schema, behorende bij **doelschema.** In het PIM model is elke klasse met een stereotype 'Domainmodel' een instantie van deze metafacade en implementeert specifiek gedrag: Het object kan van zichzelf bekijken of deze een root element is. 
	Vertegenwoordigt een <xsd:complexType> element uit een XML schema, behorende bij **bronschema.** In het PIM model is elke klasse met een stereotype 'Systemmodel' een instantie van deze facade en implementeert specifiek gedrag: Het object kan van zichzelf het volledige XSLT Xpath berekenen aan de hand van de plaats in het PIM model. 
	Vertegenwoordigt een <xsd:element> element uit een XML schema. In het PIM model is elke attribuut van een klasse een instantie van deze metafacade en implementeert extra gedrag: Het object kan o.a. de *UML multiplicity* in XSD notatie teruggeven. 
	 Vertegenwoordigt een associatie einde in een XML schema. Elk associatie einde is een instantie van deze metafacade en implementeert specifiek gedrag: Het object kan de *UML multiplicity* is XSD notatie teruggeven.
	Vertegenwoordigt een <xsd: simpleType> element in een XML schema. In het PIM model is elke klasse met een stereotype 'Enumeration' én 'XmlSchemaType' een instantie van deze metafacade. 



Vertegenwoordigt een *transformatie definitie* tussen het **bron model** en het **doel model**.
In het PIM model is elk *dependency* element met het stereotype 'XSLTTransformation' een instantie van deze metafacade.

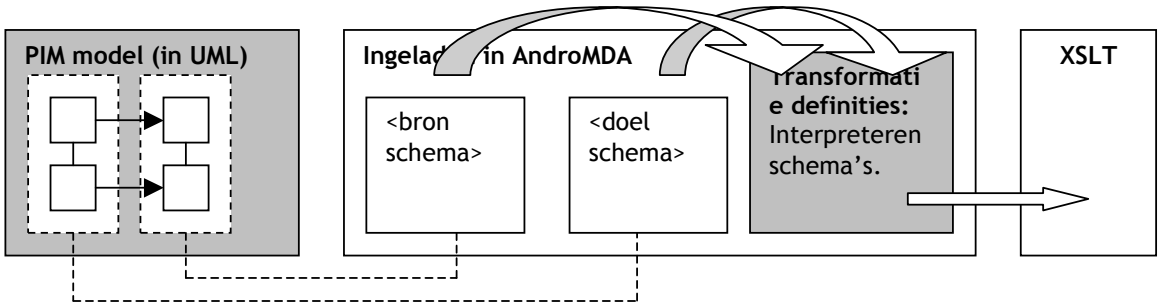
Tabel 4 – *Geïdentificeerde metafacades voor nieuwe AndroMDA cartridge.*

XSLT elementen
De tweede stap identificeert de minimale set van XSLT elementen die nodig is om aan de transformatie criteria te voldoen. De set van XSLT elementen is gevonden door voor elk voorbeeld uit tabel 3 de bijpassende XSLT template handmatig op te stellen. De onderstaande tabel toont de gevonden elementen.

XSLT element	Beschrijving
`<xsl:template match='xpath' name=''>`	Wordt gebruikt om een bepaald deel van de XML op te bouwen. Het template element is daarin te vergelijken met een functie, waarin nieuwe instructies staan.
`<xsl:apply-template select='xpath expr' >`	Wordt gebruikt om een template element aan te roepen.
`<xsl:value-of select='xpath expr'>`	Een value-of element wordt gebruikt om een waarde van een XML element te extraheren, en deze mee te nemen aan de output stream van de transformatie.
`<xsl:element name='elementname'>`	Genereert een nieuw XML element in de output stream van de transformatie.
`<xsl:text>`	Genereert in de output stream van de transformatie de exacte tekst tussen de begintag en eindtag van <xsl:text>. Wordt bijvoorbeeld gebruikt om expliciet een *spatie* aan de output stream toe te voegen.

Tabel 5 – *Benodigde XSLT elementen voor transformatie.*

Transformatie definities
De derde stap brengt de resultaten uit de vorige twee stappen bij elkaar om de mapping te leggen tussen het metamodel en de XSLT elementen. Echter, de oplettende lezer zal hebben geconstateerd dat het metamodel is opgesteld aan de hand van het XML schema, terwijl de uiteindelijke gegenereerde code XSLT moet zijn! Dit is juist. Het PIM model bestaat ook alleen uit twee XML schema's. Een vergelijking van de twee schema's uit het model resulteert vervolgens in een XSLT template. Figuur 12 toont in schema vorm dit proces.



Figuur 12 – *Procesbeschrijving: van XML schema's in het PIM model, naar XSLT.*

Tabel 6 beschrijft de transformatie definities in natuurlijke taal.

Metafacade	XSLT element
	`<xsl:template match='xpath' name=''>`
▪ Voor elk *SystemModelComplexType* uit de PIM wordt een *template* element gegenereerd. De *SystemModelComplexType* moet een transformatie definitie (*dependency*, zie tabel 4*)* hebben met een *DomainModelComplexType*, anders wordt deze genegeerd. ▪ Het *match* attribuut krijgt als waarde de naam van het *SystemModelComplexType* element. ▪ Het *name* attribuut krijgt als waarde de naam van het *DomainModelComplexType* element, aan de andere kant van de *dependency*.	

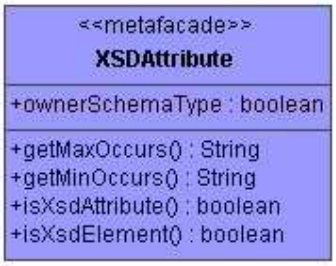
	`<xsl:template match='/'>`
	`  <xsl:element name='name'>`
	`  </xsl:element>`
	`</xsl:template>`

▪ Voor elk *DomainModelComplexType* wordt er een *element* element gegenereerd. ▪ Alleen voor de *root* wordt een `<xsl:template match='/'>` element gegenereerd, zodat de verwerking van de XSLT template op dit punt start.	
	`<xsl:apply-template select='xpath'>`
▪ Bij het genereren van elk *DomainModelComplexType* wordt er gecontroleerd of er een transformatie definitie (*dependency* element) aanwezig is vanaf een *SystemModelComplexType*. Zo ja, dan wordt een *apply-template* element gegenereerd. ▪ Het *select* attribuut krijgt als waarde de naam van het *SystemModelComplexType*. Deze refereert naar het *template* element (zie boven).	
`{@xslt.transformation}` (zie tabel 7)	`<xsl:element name='elementname'>`
▪ Voor elk `<xsl:template>` element die wordt gegenereerd (alleen als er een *dependency* tussen een *SystemModelComplexType* en *DomainModelComplexType* staat) worden de bijbehorende `<xsl:element>` elementen voor de template gegenereerd.	

- Aan de hand van de transformatie definitie uit de {@xslt.transformation} tagged value (zie tabel 7) worden de <xsl:element> elementen gegenereerd, waarbij de waarde voor het *name* attribuut uit de *doelkolom* wordt gehaald van de tagged value.

{@xslt.transformation} (zie tabel 7)	<xsl:value-of select='xpath expr'>

- Voor elke <xsl:element> element, aan de hand van de tagged value, wordt ook elk <xsl:value-of> element gegenereerd. De waarde van het *select* attribuut is afkomstig uit de *bronkolom* van de tagged value.

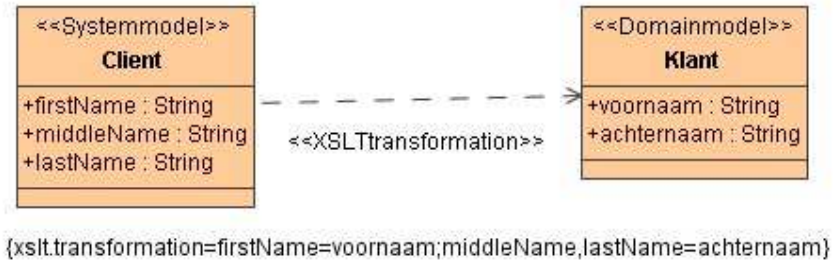
	<xsl:element name='elementname'>

- Voor elk attribuut van een klasse (type *DomainModelComplexType*) wordt <xsl:element> element gegenereerd, op voorwaarde dat deze niet in de transformatie definities voorkomt (dus dubbel voorkomt).

Tabel 6 – *Transformatie definities van metafacades naar XSLT.*

UML profiel
Met het ontwerpen van een UML profiel is er rekening gehouden met de gestelde eisen. Stereotypes geven duidelijk de classificatie aan [Berner1999] en vormen een uitbreiding op de standaard UML notatie, om de nieuwe (domein specifieke) oplossing te ondersteunen. De opgestelde notatie is tevens voorgelegd aan ontwikkelaars ter validatie. Het profiel bevat de volgende elementen:

Stereotype / Tagged Value	Beschrijving
<<domainmodel>>	Groep: Class. Elke entiteit die bij het domein model hoort.
<<systemmodel>>	Groep: Class. Elke entiteit die bij een technisch / systeem model hoort.
<<XSLTransformation>>	Groep: Relationship. Elke **dependency** tussen een domein model en een systeem model krijgt een stereotype XSLTransformation.
{@xslt.transformation}	Beschrijft de semantiek van de transformatie. Deze tagged value geeft aan welke kolommen uit het bron model worden getransformeerd naar het doel model. De syntax is als volgt: [bronveld]=[doelveld][;][bronveld][,][bronveld]=[doelveld] Voorbeeld: • firstName=voorNaam;middleName,lastName=achterNaam • zipcode,street,number=address



De transformatie tussen twee modellen in XML is als volgt:

```
<Client>                          <Klant>
 <firstName>Peter</firstName>       <voornaam>Peter</voornaam>
 <middleName>de</middleName>      <achternaam>de
 <lastName>Jong</lastName>        Jong</achternaam>
</Client>                         </Klant>
```

Tabel 7 – *UML profiel voor het PIM model.*

Velocity templates
Zoals in hoofdstuk 4.1 beschreven, maakt de architectuur van AndroMDA gebruik van Java Velocity [JavaVelocity]. Velocity is een op Java gebaseerde templating engine waarmee het mogelijk is Java objecten in een template uit te lezen en om te zetten in ASCI tekst. Deze tekst kan XML, PHP, Java code, etc zijn (zie tabel 8).

Java object	Velocity template	Output

`Persoon.setName("pieter")`	`<person>`
	`<name>$persoon.name</name>`
	`</person>`
		`<person>`
		`<name>pieter</name>`
		`</person>`

`Persoon.setName("Pieter")`	`Public class $persoon.name`
	`{`
	
	`}`
		`Public class Pieter`
		`{`
		
		`}`

Tabel 8 – *Van object naar ASCI tekst in de Velocity template engine*

De template heeft een eigen imperatieve taal: De *Velocity Template Language* (VTL). De taal is een eenvoudige template taal en is geschikt om variabel referenties uit Java objecten te lezen (zie tabel 8) en beheerst een beperkt aantal 'loop' structuren en condities. De VTL taal heeft geen uitgebreide set van helpfuncties en kan geen gebruik maken van object structuren.
De template taal bevat voor AndroMDA de transformatie definitie: In AndroMDA wordt het UML diagram ingelezen en als Java object beschikbaar gesteld aan de Velocity template, die onderdeel is van de cartridge. De template genereert dan de Java code. Voor de realisatie van de nieuwe cartridge is er ook een Velocity template ontwikkeld. De 'beschrijvende' transformatie definities uit de vorige paragraaf zijn concreet als VTL code geïmplementeerd in de template.

OCL constraints
Controle op de notatiewijze is één van de criteria. Dit zorgt er voor dat PIM ontwikkelaars op juiste wijze een UML model opstellen en daarbij worden geholpen door foutmeldingen (plus oorzaak beschrijving). Controle vindt plaats door *Object Constraint Language* (OCL) regels te plaatsen op het metamodel van het PIM model (zie bijlage E). Tijdens het inlezen van het PIM model wordt er door AndroMDA gecontroleerd of het model aan de regels voldoet (de regels van het metamodel zijn als

onderdeel van de cartridge ingeladen). Een overtreding van één of meerdere regels resulteert in een 'ModelValidationException', en stopt het codegeneratie proces.
Weliswaar is de controle met OCL niet verplicht voor het realiseren van een werkende oplossing, maar het laat – *als één van de doelstellingen van het onderzoek* – zien wat de mogelijkheden zijn met AndroMDA én de controle is gewenst voor PIM ontwikkelaars.
Voor het onderzoek is er een willekeurig aantal regels in OCL geformuleerd. Het aantal OCL regels is zeker niet volledig: De ontwikkelaar kan nog steeds een aantal elementen fout modelleren in het PIM model zonder dat dit het codegeneratie proces verstoord. Uiteraard leidt dit tot een niet-bruikbare XSLT template. Het aantal OCL regels is wél voldoende om inzicht te krijgen in de mogelijkheden van OCL.

- Elke dependency die een stereotype <<XSLTtransformation>> draagt, heeft aan het ene uiteinde een <<domainmodel>> stereotype en aan het andere einde een <<systemmodel>> stereotype.

```
context XSLTtransformation inv: sourceElement.oclIsKindOf(SystemModelComplexType)
and targetElement.oclIsKindOf(DomainModelComplexType)
```

- Normale UML associaties mogen alleen tussen entiteiten worden aangebracht van hetzelfde stereotype.

```
context SystemModelComplexType inv: associatedClasses -> notEmpty() implies
associatedClasses -> forAll(oclIsKindOf(SystemModelComplexType))
```

- Elk attribuut van een element moet een unieke naam hebben.

```
context ClassifierFacade inv: attributes -> isUnique(name)
```

- Elke aangebrachte associatie tussen twee elementen in datamodel is slechts één richting (one-way navigable). De richting geeft de parent – child relatie aan.

```
context SystemModelComplexType inv: associationEnds -> notEmpty() implies
associationEnds -> forAll(navigable xor otherEnd.navigable)
```

5. Conclusie

In dit hoofdstuk worden conclusies getrokken betreffende het onderzoek. Eerst wordt de onderzoeksvraag beantwoord waarbij is ingegaan over de mogelijkheid om transformaties uit te voeren die niet tijdens het onderzoek zijn geïmplementeerd. Daarbij is dezelfde verdeling aangehouden als in het theoretische hoofdstuk over datatransformaties. Het volgende hoofdstuk beschrijft in hoeverre AndroMDA een MDA tool is, waarna in de eindbeoordeling de toepasbaarheid van MDA in het algemeen én de AndroMDA worden besproken.

5.1. Datatransformaties met AndroMDA

Gedurende het project is er een cartridge voor AndroMDA ontwikkeld voor datatransformaties. In deze cartridge zijn - in meer of mindere mate - een aantal type transformaties geïmplementeerd. In het PIM model wordt als eerste de schemaconversie tussen twee schema's beschreven, waarna de tool een XSLT template genereert. Deze template kan de dataconversie uitvoeren. De cartridge ondersteunt momenteel een beperkt aantal datatransformaties, die zijn beschreven in hoofdstuk 4.2.3 als 'transformatie criteria'.

AndroMDA en schema-/dataconversie

De ontwikkeling van de XSLT cartridge heeft geleid tot voldoende inzichten over de mogelijkheden van AndroMDA. Hierdoor kan er een uitspraak worden gedaan over de geschiktheid van AndroMDA voor datatransformaties in het algemeen. Voor het bepalen van de geschiktheid blijken twee technische aspecten van doorslaggevend belang. Ten eerste moet het mogelijk zijn om alle informatie voor een transformatie te modelleren in het PIM model. Ten tweede moet het mogelijk zijn de PIM gegevens om te zetten naar code (zoals XSLT), die vervolgens een transformatie kan uitvoeren. Is de transformatie definitie VTL dus wel krachtig genoeg?

Puur kijkend naar de technische mogelijkheden moet geconcludeerd worden dat AndroMDA alle transformaties mogelijk zou kunnen maken. Er zijn geen aanwijzingen gevonden waaruit blijkt dat het technisch onhaalbaar is een transformatie te beschrijven in het PIM model én daarvoor code te genereren. De transformatiecode *zelf* (zoals XSLT) zou dan de enige beperking zijn. Er zijn een aantal argumenten voor deze aanname.

- Voor zover kan worden nagegaan, kan met het UML class diagram alle structuur transformaties (tabel 1) worden gemodelleerd in het PIM model. [Bommel2005] beschrijft in een apart hoofdstuk hoe UML elementen gebruikt worden om dezelfde uitdrukkingskracht te hebben als een conceptueel schema.
- Het PIM model kan transformatie informatie opnemen d.m.v. UML extensies. De 'tagged value' extensie biedt daarbij de mogelijkheid om willekeurige tekst als string op te nemen, waardoor het mogelijk wordt de meest exotische instructies via een tagged value mee te geven. Tevens biedt AndroMDA de mogelijkheid om OCL regels op te nemen, waardoor het in theorie mogelijk zou zijn condities voor transformaties (zie tabel 2) te modelleren.
- De VTL taal bevat een basisset met instructies om alle informatie uit het PIM model te verzamelen, en kan daarmee elk willekeurig patroon van tekens genereren. Er zijn geen aanwijzingen gevonden waaruit blijkt dat AndroMDA bepaalde code niet kan genereren.

Uit de bovenstaande alinea zou geconcludeerd kunnen worden dat alle transformatie, beschreven in tabel 1 en 2, met AndroMDA uitvoerbaar zijn. Echter, door een groot aantal praktische bezwaren is het onwaarschijnlijk dat er een cartridge gerealiseerd kan worden met ondersteuning voor *alle* type transformaties.

- Het PIM model kan alleen UML extensies gebruiken voor domein specifieke informatie. Voor elke transformatie moet een UML extensie worden gemodelleerd. Tijdens het uitvoeren van de case is al gebleken dat de leesbaarheid hierdoor verslechtert. Tevens verliest het PIM model steeds meer z'n platform onafhankelijkheid. Ook zijn UML extensies voor bepaalde type transformaties niet geschikt: Bij bepaalde type transformaties zijn vaak meerdere nauwkeurige parameters nodig voor het beschrijven van de transformatie. UML Extensies als 'tagged values' zijn dan niet praktisch: Zij bieden geen grafische ondersteuning voor het correct invullen van parameters en het vereist voor elke transformatie een aparte tagged value. Type transformaties zoals wiskundige expressies, strings en datums zijn daarom minder geschikt voor in het PIM model.

- Bij de implementatie van verschillende type transformaties in de Velocity template moet er steeds rekening worden gehouden met verschillende afhankelijkheden, om uiteindelijk de correcte transformatiecode te genereren. Vooral type transformaties waar condities in voorkomen maken dit proces complex. Omdat de VTL template taal eenvoudig is (zie hoofdstuk 4.2.4.), kan er geen gebruik worden gemaakt van slimme mechanismen. Dit resulteert in een slecht beheersbare 'soep' van VTL code die de verschillende transformatie definities moeten voorstellen. Naarmate er meer type transformaties worden geïmplementeerd neemt ook de foutgevoeligheid toe, omdat de VTL taal geen ondersteuning heeft voor lokale variabelen binnen subroutines.
- AndroMDA is ook minder geschikt voor datatransformaties omdat het gebruik maakt van slechts één soort model: Het PIM model. De schema's in het PIM model kunnen qua abstractie niveau te ver weg staan van het doelschema. In de gebruikte case is het PIM model een *plat* model, terwijl de gegenereerde XSLT een *hiërarchisch* XML model moet genereren. Het genereren van de XSLT wordt daardoor extra complex met de VTL taal.

Om uiteindelijk conclusies te trekken moet nog wel rekening worden gehouden met het feit dat de structuur van XSLT relatief complex is. De XML syntax kost relatief veel markup code, is niet intuïtief (en daardoor moeilijk om te schrijven) én is van een ander abstractie niveau (zoals hierboven beschreven). Een eenvoudigere transformatie taal voor XML (zoals XPathscript) zou de VTL code waarschijnlijk simpeler maken waardoor méér type transformaties kunnen worden opgenomen zonder dat de VTL code onbeheersbaar wordt.
Rekeninghoudend met deze praktische bezwaren moet worden geconcludeerd dat de ondersteuning van transformaties met wiskundige expressies en van strings en datums (zoals beschreven in tabel 2) 'zeer complex' tot 'bijna onmogelijk' kunnen worden beschouwd. De ondersteuning van de primitieve transformaties (tabel 1) van schemastructuren is minder problematisch: De gemaakte AndroMDA cartridge biedt al beperkte ondersteuning voor dit type transformaties en kan zonder domein specifieke informatie worden gemodelleerd.

AndroMDA en programmaconversie
Voor de dataconversie is nu gebruik gemaakt van XSLT. AndroMDA voert dus *niet* zelf de dataconversie uit maar genereert slechts de middelen. AndroMDA zal dus zelf nooit als dataconversie tool ingezet kunnen worden, omdat deze is ontworpen als code generator en geen externe bron kan aanroepen (zoals een database of XML bestand) voor dataconversie.
Na de dataconversie wordt de stap programma conversie beschreven in hoofdstuk 3.3. als onderdeel van datatransformatie. Een probleem bij programma conversie is parallellisatie. Het is denkbaar dat AndroMDA ook in dit probleem een rol kan spelen omdat het PIM model op een platform onafhankelijk niveau de transformatie uitdrukt. Een specifieke cartridge kan bijvoorbeeld herschrijfregels genereren (COBOL procedures naar SQL statements). Nu al is de Hibernate cartridge in staat om DDL code te genereren voor MySQL, plus de Java code voor de communicatie met de database. Natuurlijk zullen deze concepten in een ander onderzoek getoetst moeten worden.

5.2. AndroMDA als MDA tool

MDA is op het moment van schrijven een relatief nieuwe technologie, en bij Ordina is er tot op heden één project uitgevoerd m.b.v. een MDA tool. De claim dat AndroMDA een MDA tool is, is voor een groot deel waar. AndroMDA ondersteunt verschillende aspecten van het MDA framework en voldoet aan een hoop 'wensen' die worden genoemd in [Kleppe2003].

- De tool ondersteunt transformaties van PIM model naar code. De architectuur (zie figuur 9, hoofdstuk 4.1.) is zodanig opgezet dat er gebruik wordt gemaakt van één model; het PIM model. Dit wijkt af van het MDA framework van de OMG [Kleppe2003], waarin een PSM is vertegenwoordigd. De ontwikkelaars van AndroMDA verklaren dat een PSM (als model) weinig toevoegt, en dat de platform specifieke opties als 'additional markup' worden toegevoegd aan het PIM model, door middel van stereotypes en tagged values. Tevens kunnen de metafacades in de cartridge worden gezien als het PSM: Zij vertegenwoordigen de klassen uit het PSM en kunnen specifieke operaties uitvoeren die alleen horen bij het specifieke domein. Deze 'niet-tastbare' PSM opzet zorgt er voor dat de AndroMDA software daardoor niet hoeft te beschikken

over een eigen grafische omgeving (zoals de MDA tool OptimalJ), en kunnen ontwikkelaars kiezen voor hun favoriete UML tool voor het modelleren van de PIM.
- De tool ondersteunt verschillende transformaties voor domein specifieke oplossingen. AndroMDA is in tegenstelling tot de meeste CASE tools niet beperkt tot één domein. Daarnaast stelt de tool ontwikkelaars in staat zelf een transformatie definitie (in de vorm van een cartridge) te ontwikkelen.
- De tool is bijzonder flexibel. AndroMDA biedt plugin ondersteuning voor verschillende template engines (voor het opstellen van de transformatie definitie) en verschillende modelleer talen. Daarnaast kunnen bestaande transformatie definities op gewenste punten 'overruled' worden door een eigen implementatie, zonder de volledige definitie te herschrijven. Dit noemt men 'mergeLocations'.

Ook de voordelen die ontwikkelen volgens MDA oplevert (zie hoofdstuk 1.2.3.) kunnen voor een groot deel worden bevestigd:

- [Kleppe2003] en [Thomas2004] beschrijven dat MDA ontwikkelaars op een hoger abstractie niveau software ontwikkelen en zich daardoor beter concentreren op de 'wat' vraag, i.p.v. de technische details. Dit kan resulteren in software die beter aansluit op de wensen van de business. Tijdens het project is dit voordeel in de eerste fase waargenomen. Vooral de UML activity- en usecase diagrammen zijn waardevol als communicatiemiddel met de business omdat ze duidelijk de status en functionaliteit van de software weergeven.
- Een andere reden voor de hogere software kwaliteit is het feit dat de MDA tools ontwikkelaars dwingen volgens de richtlijnen van de opgestelde architectuur te werken. AndroMDA biedt de mogelijkheid om raamwerken te genereren voor o.a. Struts, Spring en Hibernate.
- MDA zou in principe moeten bijdragen aan een hogere productiviteit. Alhoewel dit niet tijdens het onderzoek is bewezen, is wel de indruk ontstaan dat dit het geval is. Met de tool AndroMDA is men in staat een volledige J2EE 5-lagen architectuur te genereren in minder dan vier uur. De opgeleverde software bestaat uit standaard componenten (CRUD functionaliteit, standaard grafische vensters, een bepaalde set beveiliging features, etc) die door het AndroMDA team zijn gedefinieerd. In veel gevallen is handmatige interventie van de ontwikkelaar niet nodig.

De OMG heeft geen standaard eisen waar een MDA tool aan zou moeten voldoen. MDA kan worden gezien als een soort toekomstvisie, die als doel heeft het handmatige 'coderingsproces' te elimineren. En dat lijkt ambitieus: Veel tools die 'MDA compatible' zijn ondersteunen verschillende aspecten van MDA, maar ontkomen (nog) niet aan de handmatige interventie van de ontwikkelaar. Ondanks het feit dat AndroMDA niet met een 'echt' PSM model werkt kan de tool als MDA tool worden bestempeld. Weliswaar is AndroMDA – net als andere CASE tools – een code generator, het verschil zit hem in het feit dat het gebruikte model *domein onafhankelijk* is en dat een UML element in het PIM model meerdere (horizontale) abstractie kan voorstellen. Dus ongeacht het 'gemis' van het PSM is het PIM model volledig platform onafhankelijk.

5.3. Eindoordeel

Het afstudeerproject kent twee hoofddoelstellingen. De eerste doelstelling is het bieden van een oplossing voor datatransformaties bij Ordina. Het bedrijf wenst de ontwikkeling van XSLT te automatiseren en daarvoor is aangenomen dat de MDA tool AndroMDA deze automatisering kan ondersteunen. Dit is als onderzoeksvraag opgenomen in hoofdstuk 1.3.3.
De tweede doelstelling is algemener en heeft als doel een advies te geven over de bruikbaarheid van AndroMDA en MDA. De visie is MDA als hulpmiddel in te zetten bij projecten, en dus niet de verwachting te scheppen dat alle op te leveren software automatisch gegenereerd kan worden (MDA als wondermiddel). Het beantwoorden van de onderzoeksvraag zou voldoende inzichten moeten opleveren om een advies te geven aan Ordina.

Datatransformaties
Transformaties lijken geschikt voor MDA. Net als het MDA framework kent een database schema verschillende abstractie niveaus: Een conceptueel schema (zonder platform specifieke details) kan als PIM model worden gemodelleerd, en het fysieke schema vertegenwoordigd het PSM model. Om 'vervuiling' van het PIM model met teveel transformatie informatie te voorkomen zouden deze

gegevens opgenomen moeten worden in het PSM model, die dan een domein specifieke grafische omgeving biedt om alles gemakkelijk uit te drukken.
MDA zou tevens geschikt kunnen zijn op verschillende platform specifieke niveaus bij datatransformatie. Denk daarbij aan code voor programma conversie (zie hoofdstuk 3.3) of DDL code voor een specifieke database structuur. Een specifieke transformatie zou vanuit een PSM code kunnen genereren voor een dergelijk probleem.

Na het uitvoeren van de case kan worden geconcludeerd dat AndroMDA niet heel geschikt is om XSLT te genereren. De grootste nadelen zijn de vervuiling van het PIM model én de onbeheersbare VTL code, naarmate er meer type transformaties worden ondersteund. Zoals in hoofdstuk 5.1 is beschreven heeft dit deels te maken met de complexiteit van de XSLT structuur en kan een XSLT alternatief het leven van een VTL programmeur waarschijnlijk een stuk aangenamer maken. Afgezien hiervan blijft de huidige modelleer omgeving ongeschikt voor transformaties, zolang AndroMDA niet beschikt over een PSM mechanisme.

Bruikbaarheid
Zoals initieel is aangegeven, moet MDA vooral als hulpmiddel worden beschouwd. In deze opzet overtuigt de MDA tool AndroMDA. De ontwikkelaars van de MDA tool leveren cartridges voor de bekende Java frameworks, en per framework kan AndroMDA een indrukwekkende hoeveelheid standaard Java code genereren. Dit bespaart ontwikkelaars veel werk.
Dit wil niet zeggen dat ontwikkelaars niks hoeven te doen. Integendeel. Om met MDA te werken is een lange inwerkperiode nodig voor mensen die niet bekend zijn met UML, modelleren en de MDA tool zelf. Bij het genereren van code die gebaseerd is op een bepaald framework, moet de ontwikkelaar ook kennis hebben van dat betreffende framework: De ontwikkelaar moet wel weten wat hij genereert, zodat er eventueel wijzigingen kunnen worden aangebracht.
Concluderend, in het algemeen zal werken volgens MDA niet lonend zijn wanneer de tool voor één project wordt gebruikt. De investering in tijd en kennis is dan hoger dan het rendement.

Voor Ordina ligt hier zeker een kans om productiviteitswinst te halen. Ordina werkt bij veel projecten volgens bepaalde Java frameworks waarvoor AndroMDA al veel standaard code kan genereren. Omdat er veel soortgelijke projecten zijn, maakt dit de investering in tijd en kennis rendabel. Naar verwachting zal de standaard code niet direct aan de kwaliteitseisen van Ordina voldoen, maar dat hoeft geen probleem te zijn. AndroMDA biedt een aantal manieren om flexibel in te grijpen in het code generatie proces, bijvoorbeeld d.m.v. *mergeLocations*. Dit is de meest eenvoudige manier om een cartridge naar eigen wens te gebruiken binnen de organisatie. De behoefte is naar mijn inziens aanwezig, aangezien Ordina momenteel bezig is met onderzoek naar de gebruiksmogelijkheden van een nieuw Java framework, genaamd *Java Server Faces* (JSF). AndroMDA kan een leidende rol spelen bij het opstellen van de eisen en het gebruik van dit nieuwe framework voor Ordina. Tevens is het mogelijk om bij het gebruik van een ‘Ordina eigen framework’ zelf een nieuwe cartridge te ontwikkelen.

6. Resultaat

De oplossing is het resultaat van drie maanden studie naar het functioneren van de MDA tool (inclusief bijbehorende tools), het analyseren van de case en het ontwikkelen van een AndroMDA cartridge. Hierdoor is voldoende inzicht verkregen in de mogelijkheden van AndroMDA en welke ontwikkelstrategieën met AndroMDA mogelijk zijn. Aan de hand van de uitgevoerde case kan er een antwoord worden gegeven op de onderzoeksvraag en wordt er in dit document een advies gegeven over de verdere toepassingsmogelijkheden bij Ordina.
Puntsgewijs is het volgende resultaat opgeleverd:

- De nieuwe cartridge voor AndroMDA is in staat uit een PIM model een geldig XSLT bestand te genereren.
- De cartridge kan het PIM model op een viertal syntactische punten valideren d.m.v. OCL.
- De cartridge genereert voor beide typen XML schema's uit het PIM model *de XML schema's in code (XSD bestanden)*.
- Er is een UML profiel ontwikkeld die in Magic Draw geïmporteerd kan worden, ter uitbreiding van de UML notatie voor het PIM model.

Reflectie
Terugkijkend naar de ontwikkeling van de cartridge is er een aantal punten op te merken. Ten eerste is de gekozen iteratieve aanpak goed bevallen. Door te ontwikkelen in kleine stappen bleef de complexiteit beperkt en werden er snel resultaten geboekt.
Voor het testen van de oplossing is de cartridge in twee delen gesplitst: De metafacades en de Velocity/descriptor files. Hierdoor konden de delen afzonderlijk getest en gegenereerd worden. Tevens zijn er Maven scripts geschreven om de twee projectonderdelen gezamenlijk te testen. Als alternatief voor het testen had ook een speciale Maven testplugin gebruikt kunnen worden die op [AndroMDA2006] beschikbaar is. De geringe tijd en het gebrek aan documentatie heeft doen besluiten de plugin niet te gebruiken. Het leek op dat moment geen meerwaarde te hebben.
Wat betreft de gewijzigde onderzoeksvraag tijdens het project zijn er positieve en negatieve punten. Het is jammer dat er vooraf geen klein risico onderzoek is gedaan naar de omvang en complexiteit van het onderwerp 'Service Oriented Architecture' (wat wél is gedaan voor AndroMDA). Dit heeft geresulteerd in een wijziging van de opdracht, waardoor een groot deel van de literatuur niet langer bruikbaar was. Aan de andere kant is deze 'tegenvaller' goed opgevangen. Er is op tijd zelf ingegrepen en een voorstel gedaan tot wijziging. Vervolgens is er snel ingelezen in een nieuw onderwerp. De case zelf kon nog wel aan SOA gerelateerd worden en bood voldoende uitdaging om de mogelijkheden van AndroMDA te ontdekken.

6.1. Toekomstige werk

Door de geringe hoeveelheid tijd en de hoge complexiteit is er een aantal zaken niet gerealiseerd, en wordt aangeduid als 'toekomstig werk'. Zo is het in de huidige oplossing niet mogelijk een XSLT template te genereren die XML bestanden terug kan transformeren van domeinmodel naar systeemmodel. Weliswaar kan AndroMDA een nieuw XSLT bestand genereren, maar dan zal in het PIM model de twee schema's moeten worden omgedraaid. De cartridge is momenteel niet in staat twee XSLT uit één PIM model te genereren, maar deze aanpassing is naar schatting te realiseren in twee weken tijd. Deze functionaliteit heeft destijds een lage prioriteit gekregen omdat deze geen extra inzicht(en) gaf in de mogelijkheden van AndroMDA.
Tevens voldoet de cartridge niet aan een aantal transformatie criteria, die zijn aangeduid als 'uitzonderlijk'. Om de cartridge volledig in te zetten bij een klant zullen ook deze criteria moeten worden toegevoegd.
Ook is er in de laatste fase van het afstudeerproject afgezien van een soort 'cookbook': Een document waarin stapsgewijs staat uitgelegd hoe een cartridge gemaakt moet worden. In overleg met Ordina is er besloten het document alleen te schrijven wanneer men besluit AndroMDA in te zetten bij projecten.
Als laatste moet worden opgemerkt dat er sinds begin Juni een Eclipse plugin beschikbaar is op [AndroMDA2006] . Deze grafische omgeving binnen Eclipse helpt ontwikkelaars bij het gebruiken van

AndroMDA. De plugin lijkt op het eerste oog erg mooi: het aanmaken en configureren van projecten is dankzij de grafische interface veel eenvoudiger. Echter, om een beter waardeoordeel te geven zal de plugin getest moeten worden, maar omdat de plugin pas aan het einde van het project beschikbaar was, is alleen de demo op Internet bekeken.

Bibliografie

[Akehurst2002] David Akehurst, Stuart Kent, A Relational Approach to Defining Transformations in a Metamodel, Lecture Notes in Computer Science, Volume 2460, Jan 2002, Pages 243 - 258

[AndroMDA2006] The AndroMDA team, homepage of AndroMDA, available from http://www.andromda.org.

[AndroMDAforum] The AndroMDA team, user forum of AndroMDA, available from http://forum.andromda.org/.

[AndroMDAmetafacades] The AndroMDA team, metafacades project documentation, available from http://www.andromda.org .

[AndroMetaAPI] The AndroMDA team, metafacade standard API javadoc, available from http://galaxy.andromda.org/docs/andromda-metafacades-uml/apidocs/index.html.

[Belo2004] Wilfred Belo, Model Driven Architecture , Master thesis, Software Engineering UvA, 2004.

[Berner1999] Stefan Berner, Martin Glinz, Stefan Joos, A Classification of Stereotypes for Object-Oriented Modeling Languages, Lecture Notes in Computer Science, Volume 1723, Jan 1999, Pages 249 - 264

[Bommel2005] P. van Bommel, Transformation of Knowledge, Information and Data: Theory and Application. IDEA Group, 2005.

[Cleve2005] A. Cleve et al, Co-transformation in Information System Reengineering, Electronic Notes in Theoretical Computer Science 137 p. 5-15, 2005

[Ct200605] Axel Kossel, Jürgen Kuri, Software Universum.

[DBMAIN] The website of DB-MAIN, available from http://www.db-main.be

[Gogolla2003] Martin Gogolla and Arne Lindow. Transforming Data Models with UML. In Borys Omelayenko and Michel Klein, editors, Knowledge Transformation for the Semantic Web, pages 18-33. IOS Press, Amsterdam, 2003.

[Henrard2002] J. Henrard, J-M. Hick, P. Thiran, J-L. Hainaut, "Strategies for Data Reengineering," wcre, p. 0211, Ninth Working Conference on Reverse Engineering (WCRE'02), 2002.

[Hunter2003] David Hunter *et al.*, Beginning XML 2nd Edition, Wrox Press, 2004

[JavaVelocity] The Apache Jakarta project, homepage of Java Velocity, available from http://jakarta.apache.org/velocity/

[Kleppe2003] A. Kleppe, J. Warmer, and W. Bast, MDA Explained: The Model Driven Architecture-Practice and Promise, Addison-Wesley, 2003.

[Lopez2004] Pere Botella I Lopez, Xavier Franch-Gutiérrez, Josep M. Ribó-Balust, Software Process Modelling Languages Based on UML, Cepis Upgrade Vol. V, No 5, October 2004.

[Mapforce] Altova, MapForce 2006, A powerful visual data mapping and

transformation tool. Information available from http://www.altova.com/products/mapforce/data_mapping.html

[Maven1.x] Apache Maven project, homepag of Maven 1.x, available from http://maven.apache.org/maven-1.x/

[MagicDraw] No Magic, home of MagicDraw, available from http://www.magicdraw.com/

[Meservy2005] Thomas O. Meservy, Kurt D. Fenstermacher, "Transforming Software Development: An MDA Road Map," Computer, vol. 38, no. 9, pp. 52-58, Sept., 2005.

[Mizuta2005] Sachio Mizuta, Runhe Huang, "Automation of Grid Service Code Generation with AndroMDA for GT3", aina, pp. 417-420, 19th International Conference on Advanced Information Networking and Applications (AINA'05) Volume 2 (INA,, USW,, WAMIS,, and IPv6 papers), 2005.

[Mork2006] Peter Mork, Arnon Rosenthal, Joel Korb, Ken Samuel, "Integration Workbench: Integrating Schema Integration Tools," icdew, p. 3, 22nd International Conference on Data Engineering Workshops (ICDEW'06), 2006.

[NetbeansMDR] Sun Netbeans, general information of the Metadata Repository (MDR) API, available from http://mdr.netbeans.org

[Newcomer2005] Eric Newcomer, Greg Lomow, Understanding SOA with Web Services, Addison-Wesley, 2004

[OMGMOF] OMG, general information of OMG's MetaObject Facility, available from http://www.omg.org/mof/

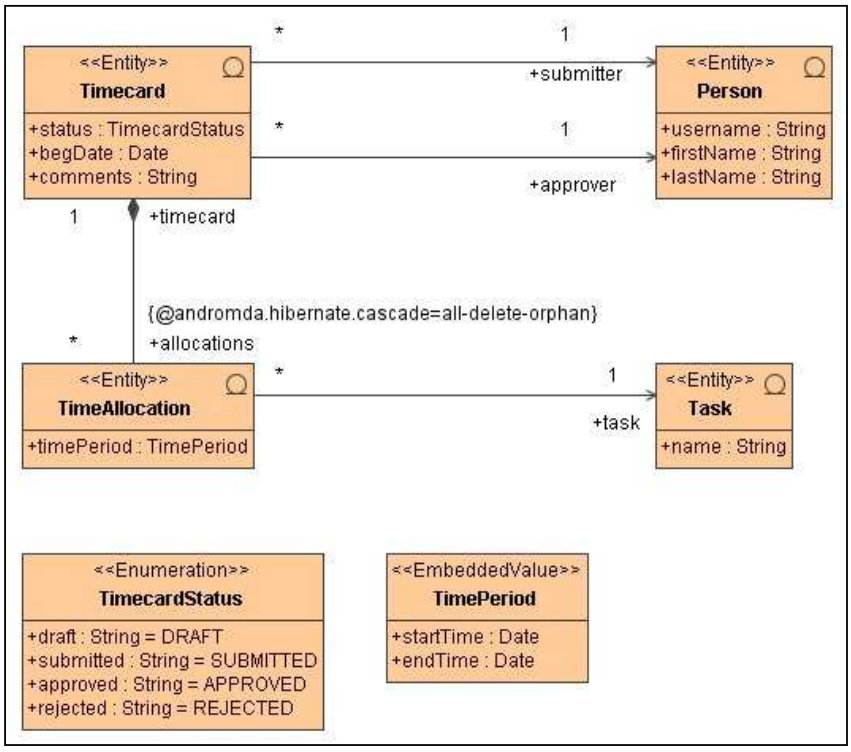
[Thomas2004] Dave Thomas, "MDA: Revenge of the Modelers or UML Utopia?," IEEE Software, vol. 21, no. 3, pp. 15-17, May/Jun, 2004.

[Schleicher2001] Schleicher, A. and Westfechtel, B. 2001. Beyond Stereotyping: Metamodeling Approaches for the UML. In Proceedings of the 34th Annual Hawaii international Conference on System Sciences (Hicss-34)-Volume 3 - Volume 3 (January 03 - 06, 2001). HICSS. IEEE Computer Society, Washington, DC, 3051.

[Warmer2003] Jos Warmer, Anneke Kleppe, The Object Constraint Language 2nd edition, Addison-Wesley, 2003

Bijlage A – UML model met AndroMDA elementen

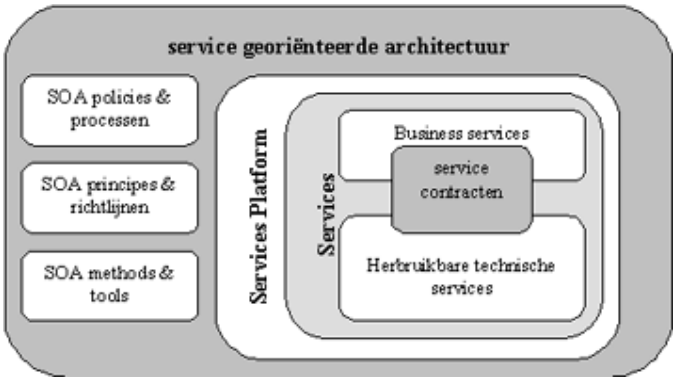
Een afbeelding van een 'marked PIM'. De modelleertool laadt een specifiek UML profiel in waardoor AndroMDA specifieke elementen beschikbaar zijn in het Class diagram (zogenaamde *UML extensions*). `<<Stereotypes>>` en `{@tagged values}` zijn daar voorbeelden van.



Bijlage B – Service Oriented Architecture

Een Service Oriented Architecture is niet letterlijk een software architectuur, zoals de naam doet vermoeden. Een SOA is een ontwerp stijl, bestaande uit verschillende aspecten die bijdragen aan de realisatie van business services. Daarnaast definieert SOA de IT infrastructuur die het mogelijk maakt deze services met elkaar te laten communiceren en data uit te wisselen, onafhankelijk van het OS of programmeertaal. Een SOA biedt o.a. de volgende voordelen:

- **Hergebruik** - De mogelijkheid om services te creëren die herbruikbaar zijn in meerdere applicaties.
- **Efficiëntie** – De mogelijkheid om snel en gemakkelijk nieuwe diensten te maken en te laten communiceren met bestaande diensten.
- **Loosely coupling** - Gebruik van een service door een andere service, waarbij alleen het contract bekend is (interface coupling).



Figuur 13– *Elementen in een SOA.*

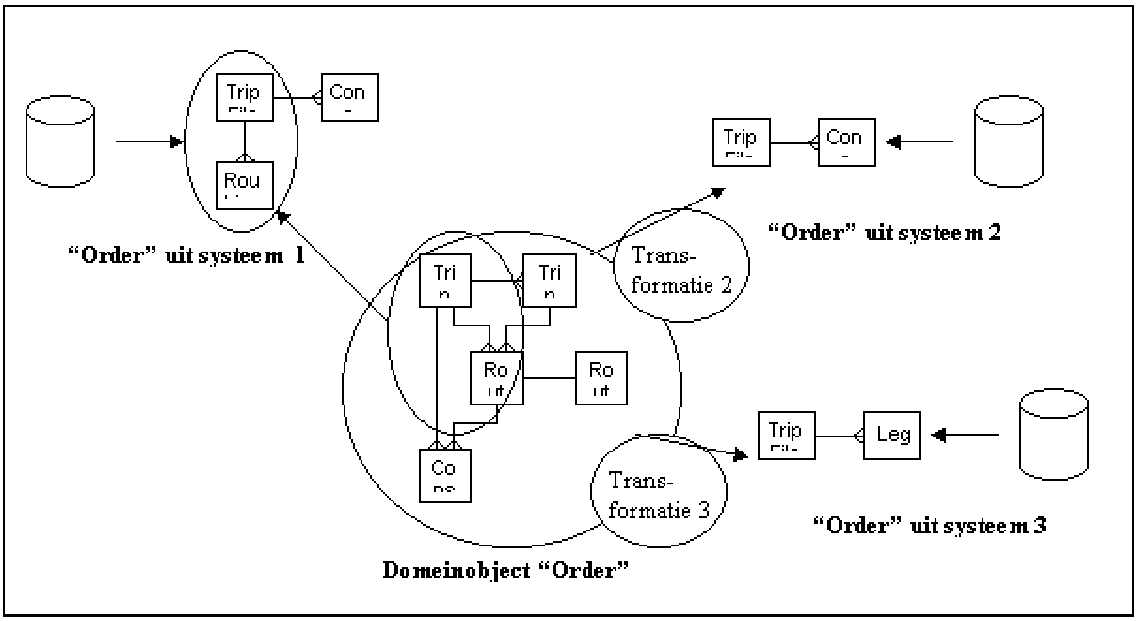
Figuur 13 toont de verschillende elementen van een service georiënteerde architectuur en is grotendeels ontleend uit [Newcomer2005].

De box *policies & processen* in figuur 13 representeert het besturingsproces van een SOA en wordt in de literatuur [2] vaak aangeduid met *governance*. Het beschrijft o.a. op papier wie welke beslissingen neemt, de verantwoordelijkheden van teams, het ontwikkel- en testproces en versiebeheer van services.
De box *principes & richtlijnen* representeert de verschillende opgestelde richtlijnen voor architecten en ontwikkelaars, voor het definiëren van services.
De box *methodes & tools* toont de verschillende tools en methodes die voor de specifieke SOA zijn geselecteerd. Denk aan design- en ontwikkeltools of bepaalde testmethodes. Ook MDA en UML tools kunnen onderdeel uitmaken van de SOA.
De technische implementatie van een SOA is in figuur 13 aangegeven als het *Service Platform*, waarbij de focus ligt op het leveren en beheren van services. Daarbij wordt er verschil gemaakt tussen business services en technische services. De eerst genoemde levert diensten die gelijk staan aan bedrijfsactiviteiten[6]. Dit is een belangrijk principe in service georiënteerde architecturen: Services worden gedefinieerd op een bepaald abstractie niveau dat correspondeert met de activiteiten in de organisatie. De technische services doen afbreuk op dit genoemde principe, maar worden wel geïdentificeerd in [Newcomer2005]. Deze services zijn niet specifiek gebonden aan een business activiteit en vormen herbruikbare services voor de business services (bijvoorbeeld logging, data transformatie, etc).
De *service contracten* in figuur 13 vormen het belangrijkste aspect van SOA. Services communiceren met elkaar door middel van een standaard gedefinieerd contract (de public interface), waarbij de implementatie van de service verborgen blijft. In het contract staat welke functionaliteit de service aanbiedt, de locatie van de service, hoe aan te roepen en onder welke voorwaarden het aanroepen moet plaatsvinden.

[6] Services worden gerealiseerd vanuit het oogpunt van de business. Om tot services te komen worden daarom eerst bedrijfsproces modellen opgesteld. Deze activiteit binnen valt buiten de scope van het project en wordt daarom niet behandeld in dit document.

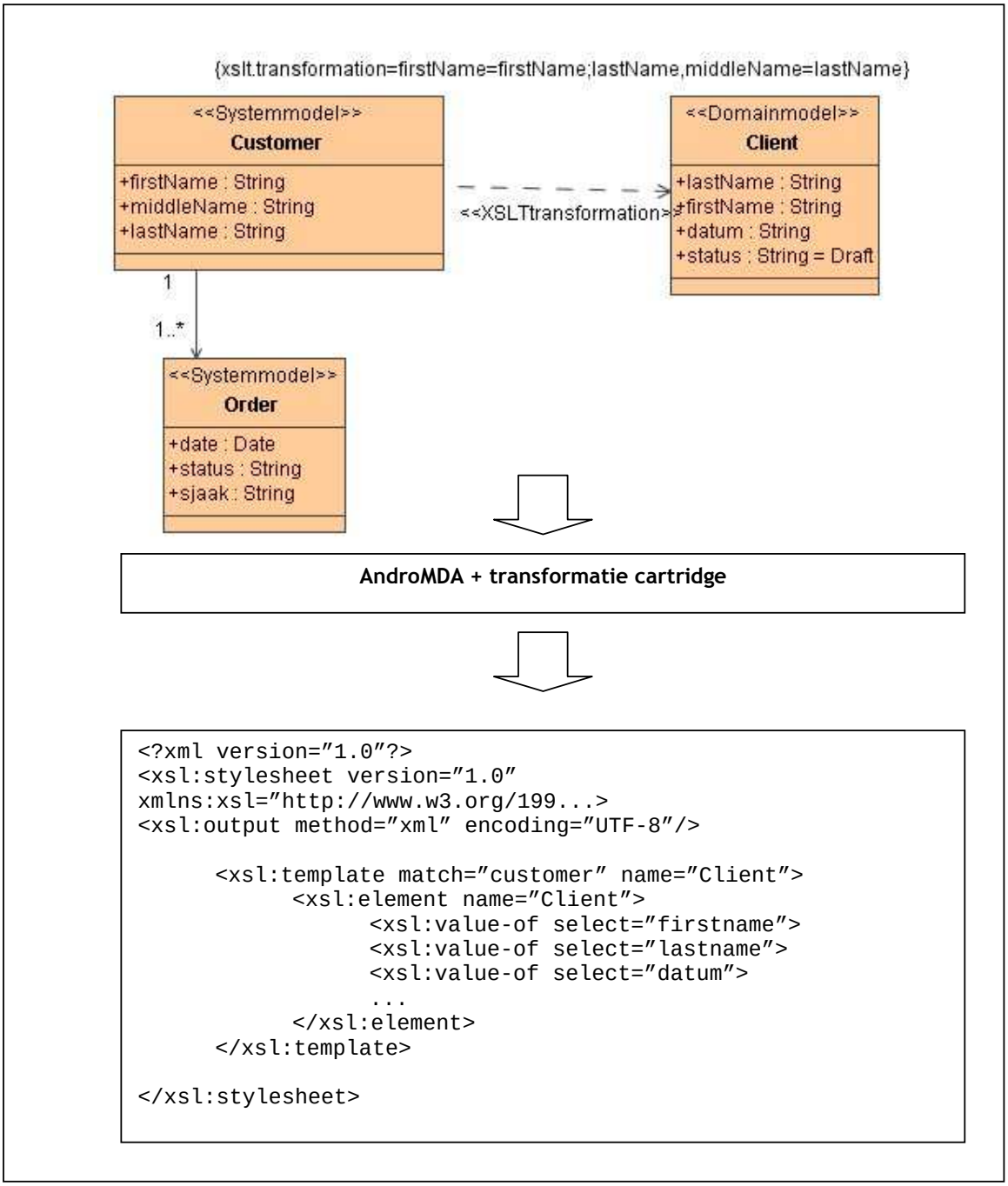
Bijlage C – Het domeinmodel in SOI

Het figuur uit bijlage C toont de functie van het domeinmodel in een Service Oriented Integration. Het uitwisselen van order gegevens tussen twee systemen gebeurt met XML berichten in een specifiek formaat. Vaak is het formaat van het bericht voor het ene systeem anders dan voor het andere systeem. Transformatie van deze XML berichten is dan vereist (bijv. door het gebruik van XSLT).
In een SOA omgeving met SOI vindt deze transformatie niet direct plaats, maar via een tussenweg. De XML berichten worden eerst getransformeerd naar het domeinmodel (een soort algemene definitie van een 'Order'), waarna het bericht naar het specifieke systeemformaat wordt getransformeerd (de bestemming).



Bijlage D – Transformaties: van PIM naar XSLT code

Ordina bouwt voor klanten SOA oplossingen waarbij gebruik wordt gemaakt van een service georiënteerde integratie (SOI). Om communicatie tussen systemen mogelijk te maken moeten de berichten van het bronsysteem eerst worden getransformeerd voordat ze voor het doelsysteem bruikbaar zijn. Voor de transformatie van berichten wordt o.a. XSLT gebruikt. De oplossing betreft het automatisch genereren van deze XSLT bestanden door zowel het bron- als doelbericht te modelleren in UML én de transformatie tussen beiden aan te geven. Voor het aangeven van deze transformatie is wel een geschikte notatie nodig, die ontwikkeld zal moeten worden. In het onderstaande figuur is een voorbeeld gegeven hoe dit proces in z'n werk gaat.



Bijlage E – UML model van de metafacades