Interfaces

De afgelopen jaren zijn er twee trends die beide tot doel hebben om software-ontwikkeling eenvoudiger te maken. Aspect Oriented Programming (AOP) is ontwikkeld om programmeren eenvoudiger en krachtiger te maken. Model Driven Development (MDD) streeft hetzelfde na, gebruikmakend van modellen. Wanneer je met beide technieken te maken krijgt, kan dit tot verwarring leiden. Dit stuk is een crash course over beide onderwerpen met de focus op de verschillen en overeenkomsten.

Model Driven Development en Aspect Oriented Programming

Twee technieken, één doel

Jos Warmer
(Partner Ordina) en
Anneke Kleppe
(Principal Consultant Capgemini)

AOP en MDA zijn ontwikkeld vanuit verschillende perspectieven: programmeren en modelleren. AOP focust op het weven van meervoudige invoer tot één uitvoer, MDD op het creëren van meervoudige uitvoer op basis van één invoer. Beide technieken stimuleren abstractie en modularisatie, zaken die voor de ontwikkeling van de IT essentieel waren en nog steeds zijn. Soms lijken we dat met zijn allen te vergeten en besteden we (te) veel aandacht aan technische zaken die hier niet toe bijdragen. AOP en MDD doen dat wel en zijn alleen daarom al de moeite waard.

Aspect Oriented Programming

Het sleutelwoord in AOP is modularisatie. Ieder programma bevat aspecten die niet specifiek zijn voor het applicatiedomein. Bekende voorbeelden hiervan zijn foutafhandeling, logging en tracing. Dit wordt in meerdere domeinen op dezelfde manier gedaan. AOP stelt dat dit soort aspecten in aparte modules beschreven kan worden. Het woord *aspect* is de naam voor zo'n module. De modules die de applicatiespecifieke code bevatten en de aspectmodules worden vervolgens met behulp van een zogenaamde *weaver* samengevoegd to één applicatie.
De applicatiespecifieke code en de aspecten kunnen op deze wijze apart ontwikkeld worden. In de applicatiespecifieke code is geen kennis nodig over welke aspecten er toegevoegd worden, de ontwikkelaars kunnen zich daarom richten op de benodigde businessfunctionaliteit. De aspect-ontwikkelaar hoeft weinig van de applicatie te weten. De kennis die nodig is voor het schrijven van aspecten gaat over *joinpoints*. Een joinpoint is een specifiek punt in de source-code van een applicatie. Voor een logging-aspect wordt bijvoorbeeld het joinpoint "method" gebruikt. Op die wijze kan bij iedere methode een extra logging-boodschap worden toegevoegd.
Joinpoints worden gedefinieerd door pointcut expressies. Het volgende voorbeeld definieert de *publicMethods* pointcut, die bestaat uit alle publieke methodes in het package *org.apache.cactus* of een van de subpackages daarvan. Voor de uitvoering van iedere methode wordt een aanroep van de Logger.entry()-methode toegevoegd, na de uitvoering een aanroep van de Logger.exit()-methode.

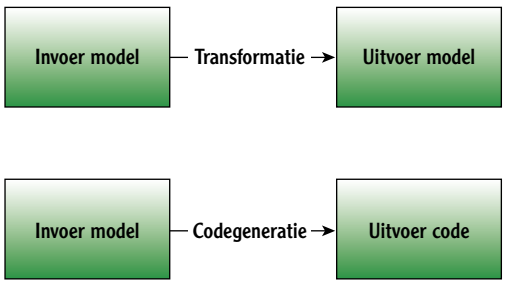
```
public aspect AutoLog{
    pointcut publicMethods() :
                execution(public * org.apache.cac-
tus..*(..));
    before() : publicMethods(){
          Logger.entry(thisJoinPoint.getSignature().
toString());
    }
```

```
    after() : publicMethods(){
          Logger.exit(thisJoinPoint.getSignature().
toString());
    }
}
```

Een *advice* is de verzameling statements die geëxecuteerd wordt wanneer een joinpoint voorkomt. In het voorbeeld is er een *before advice* en een *after advice*. Aspecten worden vaak gebruikt voor zogenaamde *crossscutting concerns*, zaken die dwars door de normale opdeling van businessfunctionaliteit gaan.
AOP komt in verschillende vormen voor, bijvoorbeeld AspectJ, Spring AOP en AscpectC++. Op AOSD.net zijn meer tools en talen voor AOP te vinden.

Model Driven Development

In MDD worden applicaties gebouwd met behulp van verschillende talen en omgevingen. Een model op hoger abstractieniveau wordt gebruikt om de business-eisen te specificeren. In dit model zijn technologische details afwezig. MDD-transformaties zetten zo een model om in lager niveau modellen, of direct naar source-code. Essentieel is dat de transformaties binnen MDD geautomatiseerd zijn. Binnen de MDA-wereld heten de hoger niveau modellen Platform Independent Model (PIM) en Platform Specific Model (PSM). Daarnaast wordt vaak gebruikgemaakt van Domain Specific Languages (DSL's).



Afbeelding 1. Model Driven Development

Transformaties komen voor in alle soorten en maten. In-model transformaties hebben hetzelfde model als invoer en uitvoer. Transformaties kunnen verschillende invoer- en uitvoermodellen hebben in dezelfde taal. Meer algemeen kunnen transformaties meerdere invoer- en uitvoermodellen hebben, potentieel allemaal in verschillende talen. Belangrijk is dat transformaties automatisch van een hoog abstractieniveau naar een lager abstractieniveau werken en in het proces allerhande technische details toevoegen.

Overeenkomsten en verschillen

Na deze heel korte introductie van de belangrijkste onderdelen van AOP en MDD is het tijd om dieper te graven. Wanneer wordt iets AOP genoemd, en wanneer MDD? Zoals zal blijken, soms een lastig te beantwoorden vraag.

Separation of concerns

Het doel van AOP en MDD is hetzelfde. Beide proberen ze complexiteit te verminderen door zogenaamde separation of concerns. In AOP ligt de nadruk op *crosscutting concerns*, in MDD ligt de nadruk op het scheiden van *technologische concerns* van de applicatielogica. Interessant is het feit dat sommige van de concerns identiek zijn. Er bestaan heel wat MDD-tools die in staat zijn om een puur businessmodel te transformeren naar een businessmodel waar persistency aan toegevoegd is. De businessmodelleur hoeft zich dan niet meer druk te maken om de technische zaken rond persistency. In een aantal AOP-voorbeelden is een apart persistency-aspect gedefinieerd. Persistency wordt vanuit AOP-perspectief gezien als een crosscutting concern, en vanuit MDD perspectief als een technologisch concern. Beide gezichtspunten zijn geldig en beide werken. Het is lastige kwestie om te beslissen welke te gebruiken.

Compile-time versus runtime

Een ander verschil tussen AOP en MDD is dat MDD de nadruk legt op een statische aanpak waarbij code gegenereerd wordt. Binnen AOP was dit in het begin ook de aanpak, de originele code wordt aangevuld met de advice code. Deze aanpak betekent dat er opnieuw code gegenereerd/geweven moet worden bij elke wijziging, en de code opnieuw gecompileerd dient te worden. Met de huidige AOP-systemen is het mogelijk bepaalde advice-acties runtime toe te voegen. Dit is een subtiel verschil, waar we later nog op terug komen. Het is overigens niet zo dat runtime of compile-time beter is. Vaak moet de nieuwe versie uitgebreid getest worden na wijziging, en daarmee wordt de snelheid van runtime-veranderingen weer teniet gedaan. In de MDD wereld begint men ook steeds vaker te praten over model-executie. Hierbij wordt het model runtime gelezen om de applicatie direct aan te sturen. De MDD-aanpak verschuift hiermee ook van compile-time naar runtime.

Modeleren versus programmeren

Een duidelijk verschil is dat AOP gericht is op programmeren, en MDD op modelleren. Zoals in mijn column *Mograms (Software Release Magazine, Januari 2008)* beschreven, is het verschil tussen programmeren en modelleren steeds minder relevant. Beide zijn beschrijvingen van

met AOP kan niet alles wat met MDD kan

een applicatie. Vandaar dat daar al de term *mogram* geïntroduceerd werd om de vereniging van beide begrippen aan te geven. Het verschil tussen programmeren en modelleren vinden we te klein om dit als een verschil tussen AOP en MDD te beschouwen.

Toegang tot het metaniveau

Zowel AOP als MDD geven toegang tot het invoermodel/programma op het metaniveau. In de MDD-wereld heeft de modeltransformatie toegang tot ieder element in het invoermodel via het metamodel van de modelleertaal. De modeltransformatie heeft ook toegang tot alle onderdelen van het uitvoermodel door middel van het metamodel van de uitvoertaal. Daarmee heeft de MDD-transformatie alle vrijheid om maximaal van alle taalconstructies (zowel in de invoer als de uitvoer) gebruik te maken.

In de AOP-wereld heeft een aspect ook toegang tot het metaniveau. Elk joinpoint definieert metaniveautoegang voor een bepaalde plek in het metamodel. In AOP is de verzameling van joinpoints voorgedefinieerd en beperkt. Vanuit een AOP-perspectief kan het metamodel daarom alleen op de voorgedefinieerde plaatsen benaderd worden. Andere delen van het metamodel, en daarmee andere delen van het invoer- of uitvoerprogramma, zijn onbereikbaar.

Hoewel beide technieken metaniveau geven is hier toch sprake van een duidelijk verschil. Uit deze vergelijking blijkt dat MDD veel krachtiger is dan AOP, vanwege de volledige toegang tot het metaniveau. Alles wat met AOP kan, is met MDD ook mogelijk, maar met AOP kan niet alles wat met MDD kan. Een kanttekening hierbij is dat AOP eenvoudiger in gebruik is doordat de joinpoints volledig voorgedefinieerd zijn.

Herstructurering

De mogelijkheden tot herstructurering zijn volledig anders in AOP en MDD. AOP kent meestal één hoofdapplicatie die met één of meer aspecten *samengeweven* kan worden. De aspecten worden allemaal in de hoofdapplicatie *gewoven*. De basisstructuur van de applicatie kan hiermee niet gewijzigd worden. Hierdoor is de basisstructuur van de resulterende applicatie altijd dezelfde als de originele applicatie. Een aspect kan wel aangeven dat er bij iedere operatie een logging boodschap dient te komen, maar een aspect kan bijvoorbeeld nooit aangeven dat de volgorde van operatie-aanroepen gewijzigd dient ter worden. AOP kan ook niet aangeven dat een operatie uit de invoer verwijderd dient te worden.

Een modeltransformatie in MDD kan de structuur van het originele model volledig wijzigen. Omdat een transformatie volledig toegang heeft tot de invoer- en uitvoermodellen, ben je vrij alles te doen wat je maar wilt. Een directe consequentie hiervan is dat MDD gebruikt kan worden om modellen te herstructureren, terwijl AOP daar niet voor gebruikt kan worden. Zo kan een klasse in een invoermodel met MDD opgesplitst worden in drie verschillende klassen in het uitvoermodel, of bijvoorbeeld in een interface en een implementatieklasse. Klassen in de invoer kunnen zelfs geheel verwijderd worden uit de uitvoer.

Een voorbeeld van herstructurering is het gebruik van MDD-transformaties om patronen toe te passen op modellen. De toepassing van veel design patterns verlangt een herstructurering van een model. Voor bijvoorbeeld een Observer/Listener-patroon dienen twee interfaces toegevoegd te worden, met daarbij de implementatie van de operaties uit de interface in de klassen zelf. Bovendien dienen de originele klassen de interfaces te implementeren. Het uitvoermodel heeft hierdoor een volledig gewijzigde structuur.

Welke taal?

Het laatste verschil waar we het hier over willen hebben, is de mogelijkheid om meerdere talen te gebruiken versus het gebruik van één taal. AOP is gericht op het gebruik binnen één programmeertaal. AspectJ is bijvoorbeeld specifiek voor Java. Hierbij kan zelfs de bytecode van een Java-applicatie aangepast worden. Voor elk AOP-systeem zijn de invoer- en uitvoertalen dezelfde taal.

MDD heeft per definitie de focus op meerdere talen. In MDD kunnen zowel de invoer- als de uitvoertalen elke taal zijn die je wenst. De enige voorwaarde is dat er een metamodel van de taal aanwezig is. Is die er nog niet, dan kan zo'n metamodel altijd zelf gemaakt worden voor elke taal die nodig is. Ook kan MDD natuurlijk binnen een en dezelfde taal gebruikt worden.

Hiermee hebben we een significant verschil gevonden. Wanneer je met meerdere talen wenst te werken, is MDD de enige optie.

Samenvoegen van AOP en MDA

Mensen zitten niet stil en we zien dan ook ontwikkelingen die AOP en MDD dichter bij elkaar brengen.

Aspecten als model of het modelleren van aspecten

Transformaties worden gedefinieerd met behulp van transformatiedefinities, die door een transformatie-engine geëxecuteerd worden. Het schrijven van een transformatie is veel werk. Ontwikkelaars zijn daarom eerst begonnen met het parametriseren van transformaties. Bij complexe parametrisering krijgen de parameters zelf de vorm van complete modellen. Op deze

manier ontstaan er modellen van persistency, modellen van security, enzovoort. MDD wordt gebruikt om deze modellen te combineren met applicatiemodellen en te transformeren naar de gewenste uitvoer. Het effect is dat een MDD-transformatie twee modellen als invoer neemt en deze, in AOP-terminologie, samenweeft. Hoewel dit op modelniveau gebeurt en op punten afwijkt, komt deze aanpak conceptueel dicht bij de AOP-aanpak..
Aspectoriëntatie wordt ook direct als concept in de modelleerwereld opgenomen. Er is zelfs een aspectgeoriënteerde modelleertaal, Theme/UML, die aspecten toevoegt aan UML. Aspectweven wordt hier modelweven.
In de AOP-wereld is er een doorlopende discussie of er wel een basisprogramma nodig is. Als alle onderdelen van een applicatie als aspect beschreven worden, is er geen noodzaak meer voor een basisprogramma. Dit gaat meer richting de MDD-aanpak.

Keuze van concerns
De kracht van de AOP- en MDD-technieken verschilt wel, maar uiteindelijk worden wel vaak dezelfde soorten concerns geïdentificeerd.

Generatie van aspectcode
Sommige MDD-omgevingen genereren niet alleen gewone code, maar ook aspectcode. De transformatie wordt hierdoor eenvoudiger en AOP-technieken worden gebruikt om verschillende getransformeerde aspecten te weven met de hoofdapplicatie. AOP en MDD worden hier in combinatie gebruikt.

Dynamische MDD
De opkomst van virtuele machines voor modellen, ofwel het rechtstreeks uitvoeren van modellen zonder tussenkomst van codegeneratie, geeft mogelijkheden om MDD dynamischer te maken, waarmee het dicht in de buurt van de dynamische varianten van AOP komt.

Toegang to het metaniveau
Het belangrijkste verschil blijven de mogelijkheden van toegang to het metaniveau. Binnen AOP is er een beperkt aantal voorgedefinieerde punten (pointcuts), maar niets houdt ons tegen om het aantal punten uit te breiden. MDD-transformaties zijn vaak lastiger te schrijven, maar het is goed mogelijk om een aantal standaard patronen in het invoermodel te vinden waar we transformaties op willen loslaten. Als we dit soort standaard patronen expliciet als apart concept benoemen, lijken deze concepten conceptueel veel op AOP-pointcuts. Op deze wijze kunnen standaard patronen eenvoudig bruikbaar gemaakt worden (het voordeel van AOP), terwijl de volledige vrijheid van transformaties voor alle andere gevallen beschikbaar is (het voordeel van MDD).

Elkaar aanvullende wegen

AOP en MDD volgen twee verschillende wegen naar hetzelfde doel. Hoewel de terminologie een vergelijking lastig maakt, blijkt bij nadere beschouwing dat de twee wegen elkaar niet uitsluiten. Er zijn meer overeenkomsten dan op je het eerste gezicht zou denken. Bij een juist gebruik kunnen beide wegen elkaar aanvullen. «