Enterprise

We weten allemaal hoe moeilijk het maken van een Java Enterprise applicatie is. Dat komt onder andere door de gelaagde architectuur waar we zo aan gehecht zijn en door het grote aanbod aan open source frameworks. Dat laatste is een zegen, maar kan ook een vloek worden, doordat er zoveel zijn om uit te kiezen. En we spreken wel over Java-applicaties, maar in werkelijkheid hebben we met veel meer talen te maken: SQL, XML, JSP, HTML. Houdt al die source files maar eens consistent!

Model Driven Development met Mod4j

Doorontwikkelen met hulp van gebruikers

Eric Jan Malotaux
is Software Architect bij Ordina. Hij is te bereiken via Eric.Jan.Malotaux@ordina.nl.

De resulterende complexiteit vertraagt de ontwikkeling van applicaties. Model Driven Development (MDD) kan helpen de complexiteit terug te dringen en daardoor de ontwikkeling te versnellen. Een model beschrijft de essentie van een oplossing, zonder allerlei implementatiedetails die het zicht daarop vertroebelen. Het is daarom veel compacter en kan veel sneller gemaakt worden. De details kunnen automatisch weer toegevoegd worden met een codegenerator. Het resultaat is gegarandeerd consistent, want de kennis van de architectuurlagen en de frameworks zit in de generator.

Toch hebben veel Java-architecten nog nachtmerries van oudere MDD-tools die gebaseerd waren op de traditionele Model Driven Architecture (MDA(®)) van de Object Management Group (OMG(™)). Die waren veel te ingewikkeld, met hun Platform Independent Model (PIM), Platform Specific Model (PSM), en de transformaties daartussen.

Het doel van Mod4j

Het doel van Mod4j is dat we sneller consistentere en kwalitatief betere Java applicaties kunnen ontwikkelen door de toepassing van Model Driven Development. Daartoe hebben we eerst een aantal samenwerkende domain-specific languages (DSL's) ontwikkeld en bijbehorende codegeneratoren speciaal gericht op administratieve webapplicaties. Die ervaring willen we gebruiken om nieuwe DSL's te ontwikkelen voor andere aspecten van applicaties. Denk daarbij aan businessrules, support voor businessprocessen en dergelijke.

Principes

Bij de toepassing van MDD hebben we geprobeerd de valkuilen uit het verleden te vermijden. In plaats van één uitgebreide modelleertaal, UML, zijn er nu vier kleine samenwerkende DSL's, die elk een aspect van de applicatie beschrijven. Het voordeel daarvan is dat ze makkelijker te leren zijn, maar ook dat ze selectief kunnen worden toegepast. En nog een nieuwigheid: de DSL's zijn tekstueel, in plaats van grafisch. Dat sluit veel beter aan bij de werkwijze van Java-ontwikkelaars. Je kunt CVS of Subversion en diff- en merge-tools gewoon blijven gebruiken. Hoewel in eerste instantie ontwikkeld door Ordina is Mod4j een open source project, gehost bij Codehaus.

Om de ontwikkeling van Mod4j te sturen zijn een aantal principes gehanteerd:

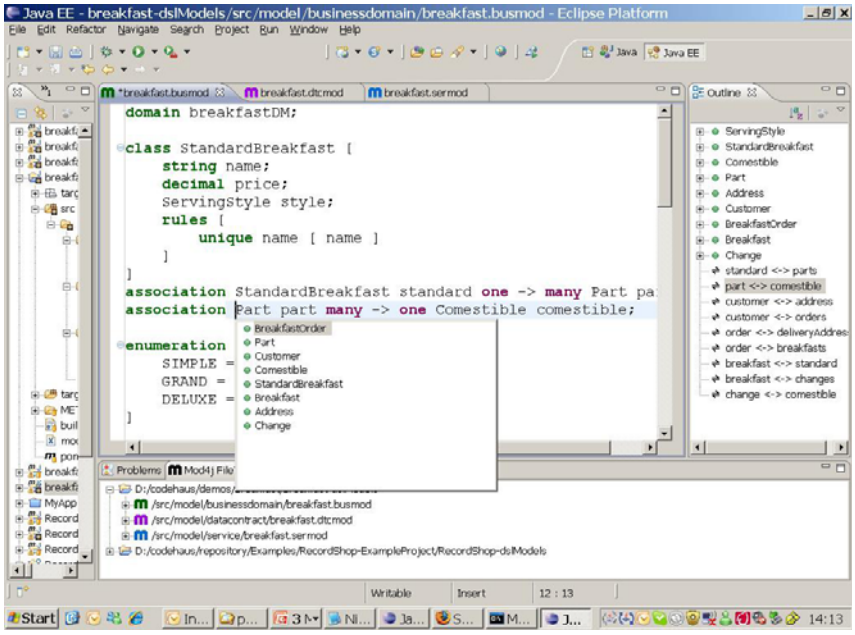
1. De DSL's zijn bedoeld voor ontwikkelaars en niet voor de gebruikers van een applicatie. Zo'n beetje eens per halfjaar voorspelt iemand dat er programmeertools zullen komen waarmee gebruikers zelf kunnen programmeren. Daar geloven wij niets van: programmeren is en blijft een vak.
2. Modelleren moet makkelijker zijn dan coderen. Zoals gezegd hebben sommigen van ons nu nog nachtmerries van traditionele MDA. Een gewone programmeur moet het kunnen.
3. Modellen worden gecombineerd met handgeschreven code. Codegeneratie is geen alles-of-niets kwestie. Beide worden gebruikt waarvoor ze het meest geschikt zijn.
4. Meerdere kleine DSL's kunnen onafhankelijk van

elkaar gebruikt worden. Als er uit een DSL niet de gewenste code gegenereerd wordt, dan moet het mogelijk zijn die niet te gebruiken en die code met de hand te schrijven, waarbij handgeschreven en gegenereerde code naast elkaar blijven bestaan. De presentatie DSL van Mod4j bijvoorbeeld is nog niet af, dus de presentatielaag moet nog met de hand geschreven worden.

5. Gegenereerde code wordt nooit handmatig veranderd. Handmatige aanpassingen blijven gescheiden van de gegenereerde code, in aparte files. De applicatie wordt gedurende de hele levenscyclus voornamelijk onderhouden op het niveau van de modellen, dus daar wordt steeds opnieuw code uit gegenereerd. Het is nauwelijks de moeite waard om de applicatie slechts éénmaal te genereren en daarna handmatig verder te ontwikkelen.
6. De gegenereerde code moet leesbaar zijn. Daar zijn verschillende redenen voor. De belangrijkste daarvan is dat de code handmatig uitbreidbaar moet zijn. Dat kan alleen als die begrijpelijk is, zodat duidelijk is hoe erop ingegrepen kan worden. Een andere reden is dat er nog geen debuggers op modelniveau bestaan. Debugging gebeurt dus op de gegenereerde code. Een derde reden is dat altijd de mogelijkheid open moet blijven dat verder onderhoud op de modellen niet mogelijk of gewenst is. Dan moet de applicatie ook met de hand onderhoudbaar zijn.
7. De gegenereerde applicatie is een normale Java (Enterprise) applicatie. Dat die met Mod4j ontwikkeld is, heeft geen gevolgen voor deployment en beheer. Daar is dus ook geen speciale opleiding of kennis voor nodig.
8. Mod4j moet flexibel genoeg zijn om mee te kunnen groeien met de voordurend veranderende omgeving. DSL's moeten aangepast kunnen worden en nieuwe DSL's toegevoegd. De codegeneratoren voor bestaande DSL's moeten aangepast of helemaal herschreven kunnen worden.

Eclipse integratie

Mod4j is gebaseerd op openArchitectureWare, dat zelf weer gebaseerd is op Eclipse Modeling Framework (EMF) en dus op Eclipse. De integratie van Mod4j in de Eclipse IDE gaat vrij ver. Zo is er een New Project Wizard, die de projectstructuur voor een nieuw Mod4j project in één keer goed neerzet. De codegeneratoren zijn gekoppeld aan het Eclipse build systeem en draaien als Eclipse builders. Een ontwikkelaar kan de automatische codegeneratie aan en uit zetten, net als de Java-compiler. Als die aan staat, wordt er code gegenereerd uit een model - en de gegenereerde Java-files gecompileerd - op het moment dat de model file gesaved wordt. De model editors hebben een outline, syntax coloring en code completion, zoals geïllustreerd in Figuur 1.



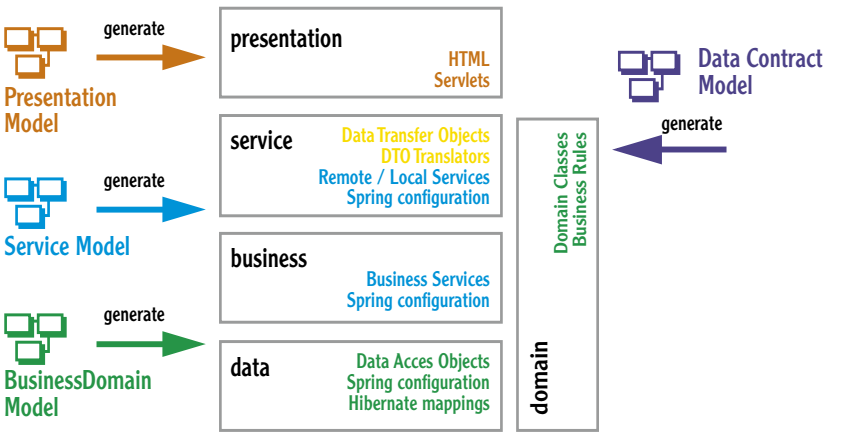
Figuur 1: Eclipse integration.

Het resultaat is een ontwikkelomgeving die nauw aansluit bij wat de meeste Java-ontwikkelaars gewend zijn.

Achitectuur

De huidige versie van Mod4j (1.0.1) bevat codegeneratoren voor één specifieke doelarchitectuur. Als je een codegenerator gaat ontwikkelen, moet je vooraf goed weten wàt je wilt genereren. De doelarchitectuur voor Mod4j is van tevoren uitgewerkt en beproefd in een handgeschreven referentieapplicatie. Die architectuur is min of meer de standaardarchitectuur die voor veel Java Enterprise applicaties gehanteerd wordt. Er is een presentatielaag, een servicelaag, een businesslaag, een datalaag en een domeinmodel dat geen laag is, omdat het door drie andere lagen gebruikt wordt. Figuur 2 illustreert de architectuur en laat zien welke onderdelen daarvan uit welk van de vier modellen gegenereerd worden.

Bij de volgende beschrijving van onderdelen van de architectuur maken we gebruik van een appli-



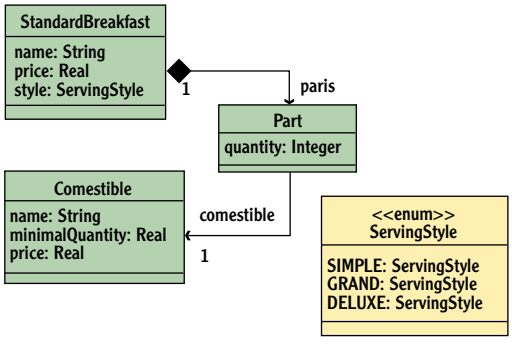
Figuur 2: Mod4j applicatie architectuur.

Met het gebruik van MDD gooien we andere technieken niet overboord

catie die ook als voorbeeld gebruikt werd in *MDA Explained: The Model Driven ArchitectureTM: Practice and Promise*: Rosa's Breakfast Service. De applicatie maakt het mogelijk voor klanten om online een ontbijt te bestellen en dat te laten bezorgen. De klant kiest uit een aantal standaard ontbijtarrangementen en kan daar nog wat extra onderdelen aan toevoegen. Figuur 3 toont een UML- model van een deel van de applicatie, namelijk de samenstelling van een standaard ontbijtarrangement.
Iedere laag in de architectuur - en het domeinmodel - wordt geïmplementeerd in een aparte module. Iedere module is tegelijkertijd een Maven-project en een Eclipse-project. Dat we nu MDD gebruiken betekent niet dat we nu ineens alle beproefde software engineering technieken overboord gooien. Dus een automatische build, onafhankelijk van Eclipse, blijft nodig, evenals Continuous Integration, waarvoor een automatische build vereist is. Tegelijk draaien de model-editors binnen Eclipse en voor de handmatige uitbreidingen in Java willen we ook graag kunnen beschikken over alle faciliteiten van een moderne IDE. De applicatie kan zowel in Eclipse als door Maven gebouwd worden.
Veel Java-architecten die gereserveerd staan tegenover MDD noemen frameworks als alternatief. Frameworks kunnen een zekere structuur bieden aan een applicatie. Dat geldt voor een gegenereerde applicatie net zo als voor een handgeschreven applicatie en Mod4j maakt dan ook gebruik van gangbare frameworks als Spring en Hibernate. Dat heeft meteen een gunstig effect op de omvang en de onderhoudbaarheid van de codegeneratoren. De Spring configuratiefiles en de Hibernate mapping files worden uit verschillende modellen gegenereerd, zoals te zien is in Figuur 2.

Business Domain DSL

Een domeinmodel vormt de kern van een Mod4j-applicatie. Domain Driven Design (DDD) is een ontwikkelparadigma dat steeds populairder wordt in de Java-wereld. Die aanpak kan goed met MDD gecombineerd worden. In het domeinmodel wordt de wereld gemodelleerd waar de applicatie over gaat, de wereld van de gebruiker. In onderstaand voorbeeld wordt een deel van het domeinmodel van Rosa's Breakfast Service getoond in de textuele syntax van de businessdomein DSL van Mod4j; hetzelfde deel als in het UML model van Figuur 3.



Figuur 3: Rosa's Breakfast Service–UML.

```
domain breakfastDomainModel;
enumeration ServingStyle [
    SIMPLE = 1;
    GRAND = 2;
    DELUXE = 3;
]
class StandardBreakfast [
    string name;
    decimal price;
    ServingStyle style;
    rules [
        unique name [ name ]
    ]
]
class Part [
    integer quantity;
]
class Comestible [
    string name;
    integer minimalQuantity;
    decimal price;
]
association StandardBreakfast standard
    one -> many Part parts;
association Part parts
    many -> one Comestible comestible;
```

De overeenkomst tussen de twee modellen is makkelijk te zien. De naam van het model, 'breakfastDomainModel', wordt in andere modellen gebruikt om aan dit model te refereren.
Het aantal files dat gegenereerd wordt uit het domeinmodel is afhankelijk van het aantal classes en businessrules dat daarin gedefiniëerd is. Voor het gegeven voorbeeld zijn het er vijfendertig. Dat zijn ten eerste twee Java classes per domein class. Volgens principe 5 wordt gegenereerde code apart gehouden van handgeschreven aanpassingen. Mod4j past daarvoor het 'Generation Gap' pattern toe, dat al in 1996 is beschreven door John Vlissides (www.research.ibm.com/designpatterns/pubs/gg.html). Alle gegenereerde code staat in een abstracte superclass en de handmatige aanpassingen komen in een concrete subclass, die éénmaal wordt gegenereerd (als die nog niet bestaat) en daarna nooit meer wordt overschreven. Dit pattern heeft als voordeel boven het vroeger veel toegepaste 'guarded blocks' pattern, dat het is gebaseerd op het voor Java ontwikkelaars bekende OO-principe van overerving.

In onze architectuur worden de Java domein classes geacht gedurende hun hele levenscyclus altijd in een geldige toestand te verkeren. Daarom hebben ze constructors met parameters voor alle verplichte properties. De constructor van de gegenereerde abstracte superclass StandardBreakfastImplBase ziet er als volgt uit:

```
public StandardBreakfastImplBase(String name,
   float price, ServingStyle style) {
    this.name = name;
    this.price = price;
    this.style = style;
    addValidators();
    validation.validate();
}
```

En die van de handgeschreven concrete subclass StandardBreakfast zo:

```
public StandardBreakfast(
String name, float price, ServingStyle style) {
          super(name, price, style);
}
```

Alle constraints worden gevalideerd bij de aanroep van iedere method, inclusief de constructor, die iets aan de toestand van het object kan veranderen. Daarvoor dient de validation.validate() aanroep in StandardBreakfastImplBase. De implementatie van de addValidators() method is als volgt:

```
private void addValidators() {
validation.addValidator(new NotNullValidator(
          StandardBreakfast.class, "name"));
validation.addValidator(new NotNullValidator(
          StandardBreakfast.class, "price"));
validation.addValidator(new NotNullValidator(
          StandardBreakfast.class, "style"));
}
```

Het veld 'validation' is een instantie van de BusinessRuleValidationSupport class die onderdeel is van een klein support frameworkje dat bij Mod4J hoort. De class is gebaseerd op de Validator interface van het Spring framework. Nieuwe validators kunnen eenvoudig worden ingeplugd, waarna ze automatisch worden uitgevoerd door de hierboven getoonde aanroep van validation.validate().

In de datalaag worden per domein class twee Java classes, twee Java interfaces en één Hibernate mapping file gegenereerd. Data access objects vormen één van de soorten objecten waarvan het interface gescheiden is van de implementatie, met als doel het verminderen van de coupling tussen classes en hun gebruikers. Aan het basis 'Generation Gap' pattern wordt daarvoor een gegenereerde superinterface en een handgeschreven subinterface toegevoegd.
Tot nu toe hebben we eenentwintig gegenereerde files beschreven. Van de overblijvende veertien zijn er zes Spring XML configuratie files, vier Maven Project Object Model (POM) XML configuratie files en één properties file. De Spring configuratie files worden steeds in paren per module gegenereerd, volgens een variant van het 'Generation Gap' pattern. Met de derde Spring configuratie file van iedere module wordt van de eerste twee een hierarchische Spring context opgebouwd en tegelijk de context van de volgende laag aangekoppeld. De POM's maken het mogelijk de hele applicatie ook buiten Eclipse te bouwen, met Maven. Nu zijn er nog vier onverklaarde files over. Eén is een Java class die een generiek Hibernate user datatype implementeert voor enumeraties. De overige drie zijn Java classes die gebruikt kunnen worden als dragers van zoekcriteria voor het aanroepen van de findByExample() method die Mod4j genereert voor iedere domein class.
De bespreking van het datacontractmodel en het servicemodel en de daaruit gegenereerde code is omwille van de ruimte weggelaten. De sourcecode van de complete voorbeeldapplicatie, waaronder ook het datacontract model en het service model, zijn te vinden in de subversion repository van Mod4j: http://svn.codehaus.org/mod4j/trunk/demos/breakfast/.

CrossX

Zoals eerder in principe 4 werd gezegd, kunnen de DSL's onafhankelijk van elkaar gebruikt worden. Toch wordt er van modellen naar andere modellen gerefereerd, niet alleen naar modellen van dezelfde DSL, maar ook naar die van andere DSL's. Om dat mogelijk te maken is er een broker ontwikkeld, CrossX genaamd, die bemiddelt tussen de modellen. Ieder model publiceert een deel van zijn informatie naar de broker, waar andere modellen die informatie kunnen vinden. Die hoeven daardoor niet te weten waar die informatie vandaan komt, maar hebben er voldoende aan te weten dat een element van het vereiste type bestaat. CrossX is de component die het mogelijk maakt een applicatie modulair te modelleren. Om code te genereren uit één model is het niet nodig andere modellen waaraan in dat model gerefereerd wordt ook in memory te laden. Dit maakt het proces van codegeneratie schaalbaar, vergelijkbaar met het compileren van Java-code. Codegeneratie gaat zó snel, dat het mogelijk is dit automatisch te doen bij het opslaan van een model.
De CrossX technologie wordt door ons ingebracht in het Eclipse platform zelf, omdat daar grote behoefte aan is.

Toekomst

Nu we eenmaal een werkende model-driven omgeving hebben, borrelen er allerlei ideeën bij ons op voor nieuwe mogelijkheden. Voorbeelden:

- Nieuwe codegeneratoren om met de bestaande DSL's SOA-componenten te genereren. De uitdaging daar is dat componenten met webservices moeten kunnen communiceren op basis van een canonical datamodel, dat vooraf gegeven is in de vorm van een set van XSD's en WSDL's.
- We zouden de concepten uit het populaire boek 'Domain Driven Design' van Eric Evans in de bestaande DSL's kunnen opnemen. Dat zou het implementeren van zijn concepten een stuk makkelijker maken
- Nieuwe DSL's om bijvoorbeeld businessrules of businessprocessen te modelleren.

Als open source project hangt de richting waarin Mod4j zich zal ontwikkelen mede af van de inbreng van de gebruikers. We hopen dat velen van jullie Mod4j zullen gaan gebruiken en de ontwikkeling ervan mede vormgeven. «

Referenties

MDA Explained: The Model Driven ArchitectureTM: Practice and Promise. Anneke Kleppe, Jos Warmer, and Wim Bast. 2003. Addison-Wesley.

Domain-driven design: tackling complexity in the heart of software. Eric Evans. 2004. Addison-Wesley.

http://www.mod4j.org/
http://www.openarchitectureware.org/
http://www.research.ibm.com/designpatterns/pubs/gg.html