TOOLS

Met de economie in een dal herzien veel organisaties hun doelstellingen voor hun software development. Lange termijn-projecten maken plaats voor kortere, meer agile projecten met haalbaardere doelen en kortere ontwikkeltijden. Om deze doelen te kunnen halen wordt gestreefd naar lage kosten en hoge productiviteit. Model driven development is een technologie die daarbij van nut kan zijn. Hoe model driven development uitermate pragmatisch is in te zetten in projecten, leest u in een serie van twee artikelen. In dit tweede deel bekijken we het gebruik van de codegenerator Tobago MDA.

Pragmatisch model driven development

Tobago MDA versnelt projecten

Sander Hoogendoorn is principal technology officer bij Capgemini.

Rody Middelkoop is senior technology consultant bij Avisi.

Robert de Wolff is senior technology consultant bij Capgemini.

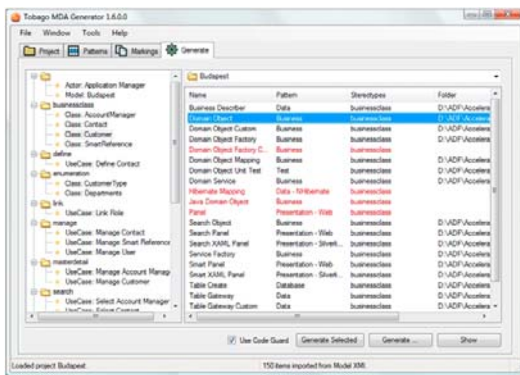
Model driven development kan een rol spelen is heel diverse projecten. Denk maar aan Java, .NET, SharePoint of zelfs SAP of BI projecten. In de regel beslist de klant over de te gebruiken modelleer- of ontwikkelomgevingen. Dan werkt het gebruik van een intermediaire codegenerator het best – een gereedschap dat het model inleest vanuit een standaardformaat (zoals XMI) vanuit de modelleeromgeving, en code genereert naar de ontwikkelomgeving van keuze. Vaak gebruiken we Enterprise Architect of een van de Rational modelleergereedschappen in combinatie met .NET en Visual Studio of Java in combinatie met Eclipse. Tobago MDA is zo'n codegenerator. Dit pragmatisch gereedschap is ontwikkeld door het core team van Capgemini's agile Accelerated Delivery Platform en is vrij beschikbaar voor iedereen die zich registreert op de wiki van het platform (www.accelerateddeliveryplatform.com).

Tobago MDA voert net als andere intermediaire model driven development tools een aantal acties uit:

- Define project. Wanneer je een nieuw project start moet je dit eerst definiëren.
- Define patterns. Een pattern is een collectie (platte) tekstgebaseerde bestanden die samen kunnen worden gebruikt om code of andere deliverables te genereren.

Zo'n pattern bestaat uit een patroondefinitie, de hoofdtemplate en mogelijk een kleine directorystructuur met de benodigde aanvullende templates. Ieder patroon wordt gebruikt om één type deliverable te genereren, bijvoorbeeld een domeinobject in .NET, een domeinobject in Java, een Excel-spreadsheet met een smart use case schatting of een webpagina gebaseerd op een enkele use case.
Het spreekt voor zich dat de verzameling patterns die beschikbaar is voor Tobago MDA snel groeit vanuit de projecten waarop het wordt toegepast. Op het ogenblik bevat deze collectie meer dan honderd verschillende patronen die zijn gericht op verschillende programmeertalen en frameworks en zelfs Worddocumenten voor het testen en Excel-spreadsheets voor het maken van schattingen.

- Import model. De volgende stap is het exporteren van het model uit de (UML) modelleeromgeving in het bij UML behorende standaardformaat XMI. Dit model wordt vervolgens in Tobago MDA geïmporteerd.
- Generate deliverable. Last but not least bindt de codegenerator een set van patronen aan een (set van) modelelement(en). Om te definiëren welke patronen werken op welke modelelementen worden stereotypes toegevoegd. Als een domeinobject bijvoorbeeld het stereotype 'domainobject' krijgt, gelden er een (groot) aantal patronen, inclusief .NET en Java domeinklassen, een domeinobject unit test, een table gateway, zowel Hibernate als nHibernate configuratiebestanden en zelfs een SharePoint webpart.



Figuur 1.

Hoewel deze stappen het basisproces van het genereren van deliverables met Tobago MDA goed beschrijven, is er natuurlijk nog veel meer te vertellen. Zomaar een paar handige aantekeningen:

- Combine model elements. Verschillende elementen van het model kunnen worden gecombineerd om nog krachtigere structuren te genereren; twee verbonden domeinobjecten genereren bijvoorbeeld relaties in code en ook in de database.
- Combine types of model elements. Het combineren van verschillende typen modelelementen biedt nog meer mogelijkheden. Normaal gesproken voegen we domeinobjecten toe als properties aan een smart use case en stereotyperen deze properties bijvoorbeeld met 'manage', 'search' 'master', 'detail' of 'service'.
- Door een use case 'Search Customer' te decoreren met een property van het type 'Customer' en het stereotype 'search' kunnen we een volledig functionele webpagina genereren.
- Remember location. Een prettige functie van Tobago MDA is dat de tool de directory waar de gegenereerde code staat onthoudt zodat makkelijk hetzelfde pattern kan worden gebruikt.

Het model veranderen zonder codeverlies

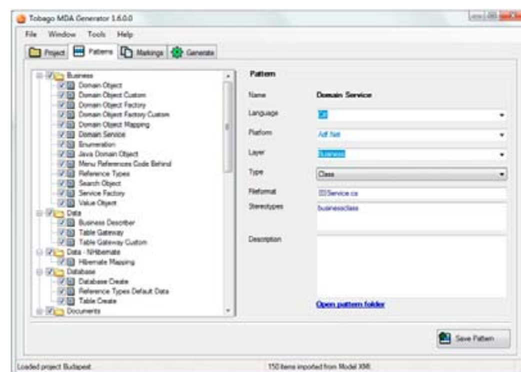
Een van de belangrijke eigenschappen van codegeneratoren is de mogelijkheid om opnieuw te genereren, zonder dat de inmiddels met de hand toegevoegde code verloren gaat. Het onderliggende model verandert immers gedurende een project door nieuwe inzichten. In veel gevallen is nu een nieuwe versie van de gegenereerde code nodig. Er kunnen verschillende mechanismen worden toegepast om hergebruik van code toe te staan. In principe gebruiken de meeste tools één of meer van de volgende technieken:

- Partial classes. In .NET is het mogelijk partial classes te definiëren. Nu kan de gegenereerde code in één bestand worden geplaatst en de handmatige code in een ander bestand, terwijl beide bestanden bij dezelfde class horen.
- Inherited classes. In Java, waar partial classes niet beschikbaar zijn, kan overerving worden toegepast. De handmatige code wordt nu toegevoegd aan een class die direct van een gegenereerde basisklasse erft.
- Marking code. Een derde optie die gebruikt kan worden in alle programmeertalen het plaatsen van handgeschreven code tussen speciale 'markers'. Als een bepaalde class opnieuw wordt gegenereerd bewaart de codegenerator alle code tussen deze 'markers' en voegt de oude code samen met de nieuw genereerde code.

Merk wel op dat geen van deze technieken waterdicht is. Er zijn altijd situaties waarbij een bepaalde techniek niet toegepast kan worden. De genoemde methoden werken bijvoorbeeld niet als je deliverable een Worddocument of een Excel-spreadsheet is.

Het gebruik van patterns met Tobago MDA

Er zijn diverse (model driven) codegeneratietools op de markt. Een groot aantal daarvan is erg complex in het gebruik. Bij het ontwerp van Tobago MDA was ons primaire doel om eenvoud in gebruik te bewerkstelligen. De basistemplate in elk pattern is hier eenvoudigweg gebaseerd op bestaande codevoorbeelden. In principe doen we dit door een kleine domeinspecifieke taal (DSL) voor het genereren van de code te combineren met een slimme directorystructuur.



Figuur 2.

Deze DSL bevat tags die de properties direct gebruik maken van het metamodel van het geïmporteerde model. Een object 'Use Case' heeft in het metamodel een property 'Name'. In de template wordt de tag '$UseCase.Name$' gebruikt. Als Tobago MDA op basis van deze template code genereert met een use case uit het model, wordt deze tag vervangen door de echte naam van de use case, bijvoorbeeld 'Search Customer'.

Naast de tags voegt Tobago's domeinspecifieke taal een aantal operaties toe. Sommige daarvan, zoals

Je kunt code opnieuw genereren zonder dat de met de hand toegevoegde code verloren gaat.

Er zijn maar heel weinig beperkingen aan wat je met deze technologie kunt doen.

'$UseCase.Name.Trim$' kunnen direct aan de tags worden toegevoegd. In dit geval worden spaties uit van de use case verwijderd. Dit kan bijvoorbeeld nodig zijn om een geldige classname voor de use case te creëren bij het genereren van code.

Andere operaties hebben tags van het metamodel als argumenten. Zulke operaties worden uitgevoerd door Tobago MDA als op basis van een modelelement en een template code wordt gegenereerd. Een voorbeeld daarvan is de 'Tobago.Loop()' operatie. Deze operatie loopt door de properties van een klasse en zoekt naar vergelijkbare templates in de subdirectories, zoals bijvoorbeeld '\DAOs' en '\Attributes'. De gevonden templates worden toegevoegd aan de gegenereerde code. De resulterende code wordt ingevoegd in de hoofdtemplate.

Hier is een voorbeeld van zo'n template te zien. In dit geval het genereren van een Java service implementatie voor een use case:

```
package nl.$UseCase.Model.Name.Lower$.service;

import javax.ejb.EJB;
import javax.ejb.Stateless;
import nl.$UseCase.Model.Name.Lower$.model.*;
import nl.$UseCase.Model.Name.Lower$.dao.*;
import org.jboss.annotation.ejb.LocalBinding;
import java.util.List;

@Stateless
@LocalBinding(jndiBinding = "$UseCase.Model.Name.
Lower$/$UseCase.Name.Trim$Service")
public class $UseCase.Name.Trim$ServiceImpl implements
$UseCase.Name.Trim$Service {

<Tobago.Loop(UseCase.Attributes, "DAOs")>

<Tobago.Loop(UseCase.Attributes, "Attributes")>
}
```

Hieronder een tweede voorbeeld. Deze voorbeeldtemplate werkt op een enkele property van een klasse in C#. Deze template is typerend voor het gebruik van 'Tobago.Loop'. Er kan bijvoorbeeld voor alle properties van een class mee worden gegenereerd.

```
<Tobago.If($Attribute.IsNullable$,"", "[NonEmpty]")>
public $Attribute.Type$ $Attribute.Name$
{
     get
     {
       return state.GetValue<$Attribute.Type$>($Attribute.
Owner$Describer.$Attribute.Name$);
     }
     set
     {
     state.SetValue($Attribute.Owner$Describer.$Attribute.
Name$, value);
     }
}
```

Wanneer past een pragmatisch modelgedreven scenario?

Bovengenoemde benadering is uitermate pragmatisch. Het is niet moeilijk om nieuwe patterns te creëren, zelfs als je daar weinig ervaring mee hebt. Begin met bestaande code die overeenkomt met het soort deliverables dat moet worden gegenereerd. Daarna kijk je welke delen van deze code (of andere deliverables) vervangen moeten worden door tags. Vervolgens haal je de daar terugkerende onderdelen uit, voeg je properties en operaties toe op basis van Tobago's DSL en verplaatst de nieuwe subtemplates naar subdirectories. Nu is alles klaar om de code te genereren en toe te voegen aan je applicatie. Dit proces kost hooguit een kwartier en kan vervolgens eindeloos toegepast voor elementen in je softwarearchitectuur.

Deze manier van codegeneratie is snel en eenvoudig toe te passen, gegeven een aantal randvoorwaarden:

- UML. Je moet weten hoe je moet modelleren in UML; welke modelleertechnieken je moet gebruiken en hoe je ze moet gebruiken. In de voorbeelden gebruiken en combineren we de twee meest eenvoudige basistechnieken die er zijn: het use case diagram en het klassendiagram.
- Softwarearchitectuur. Je moet weten welke code je wilt genereren. Dit hangt sterk af van de onderliggende softwarearchitectuur; uit welke lagen deze bestaat; welke typen elementen er gebruikt worden in deze lagen. In het algemeen zijn deze elementen getransformeerd in patterns. Denk daarbij aan domeinobjecten, repositories, factories, table create statements, controllers, views. Afgezien van de codegeneratie is een degelijke softwarearchitectuur in ieder geval een must om goede en onderhoudbare software te schrijven.
- Frameworks. Om het ontwikkelproces nog verder te versnellen is het aan te bevelen standaard- of open source frameworks te gebruiken, zoals Spring, Enterprise Library, ADF, Castle of nHibernate. Let wel op dat je de gebruikte frameworks afstemt op je softwarearchitectuur en niet andersom.

Gegeven deze randvoorwaarden zijn er maar weinig beperkingen aan wat je met deze technologie kunt doen. Of je nu een Excel-spreadsheet met een schatting op basis van use cases en hun complexiteit uitgenereert, of een Worddocument met testscenario's, complete functionele domeinlagen of op actoren gebaseerde menu's genereert, het is eenvoudig te realiseren. Een voorbeeld.

Java genereren

Avisi in Arnhem, een bedrijf dat voornamelijk in Java ontwikkelt, gebruikt Tobago bij de ontwikkeling van een human resources (HR) applicatie om bevoegdheden, werknemers, administratieve overeenkomsten met werknemers en dergelijke te beheren. De applicatie is grotendeels gebouwd uit een Java open source stack en bevat de volgende componenten: Freemarker, Sitemesh, Struts2/Xwork,

Spring, JPA/Hibernate, MySQL en JBoss. De applicatie is gegenereerd met Tobago en vervolgens gedeployed met JBoss en kan direct door de gebruiker worden ingezet.

Het hieronder getoonde formulier is gegenereerd uit de use case 'Add Employee' in het model te combineren met het domeinobject 'Employee'.

Het veld 'Gender' is een combobox die gevuld is door gebruik te maken van een enumeratie in het model. De andere combobox is automatisch gevuld, omdat het veld 'Country' gemodelleerd is als een smart reference op het domeinobject 'Employee'. Alle velden zijn gedecoreerd met reguliere expressies, die zijn als validaties zijn ingevoerd in het UML-model.

In het tweede formulier (Figuur 4) kunnen aan een werknemer bevoegdheden worden toegevoegd en kan hij cursussen volgen. Dit formulier bouwt voort op de relatie tussen 'Employee', 'Competence' en 'Training' in het domeinmodel.

Figuur 3.

Figuur 4.

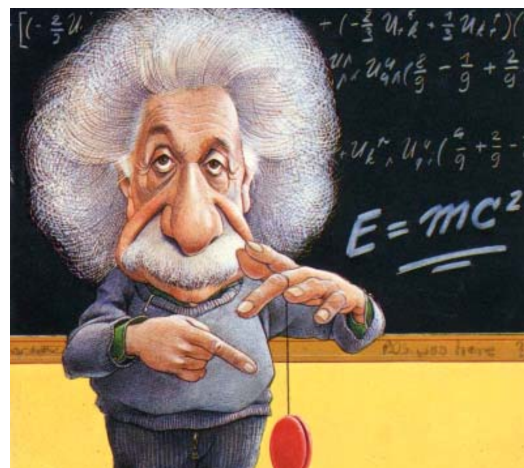
Pragmatische aanpak werkt het beste

In onze beleving kan een erg pragmatische aanpak van model driven development zoals hier beschreven erg effectief zijn. Hoewel er vele alternatieve strategieën voor model driven development bestaan, vinden wij dat een pragmatische aanpak het beste werkt. Hoe theoretischer een aanpak is, hoe moeilijker het is om die in de dagelijkse praktijk te gebruiken. Wij maken vooral gebruik van smart use cases en het domeinmodel en kleden dat aan door deze te decoreren met stereotypes.

Ondanks dat enkele fabrikanten codegeneratie faciliteren in hun modelleer- of ontwikkeltools, hebben we er bewust voor gekozen om een onafhankelijke, intermediaire tool zoals Tobago MDA te gebruiken en ons zo te richten op meerdere platforms met dezelfde strategie. Alle patterns die we hebben ontworpen voor Tobago MDA zijn opgezet vanuit bestaande code en dus met al bewezen technologie. Deze patterns gebruiken om code te genereren verhoogt de kwaliteit van de software.

Verwacht echter niet dat model driven development perfect is, maar dat hoeft ook niet. Het is zeker geen silver bullet. Code en andere deliverables worden gegenereerd om vervelende, repetitief werk, zoals het ontwikkelen van webformulieren, panels, domeinvoorwerpen, tabellen, service interfaces en zelfs documentatie te vermijden. Dit stelt ontwikkelaars in staat zich te concentreren op business logic of de fijnafstelling van de user interface.

Deze aanpak bespaart tijd en moeite in een gevarieerd aantal scenario's en ontwikkelomgevingen. Ze is succesvol gebleken in het (gedeeltelijk) genereren van oplossingen voor verschillende soorten projecten, met inbegrip van ASP.Net, Silverlight, SharePoint, Visual Basic Windows Forms, Java, service georiënteerde applicaties (.Net en SAP) en zelfs backend functionaliteit. Deze aanpak voldoet aan een van Albert Einstein's beroemde citaten: 'alles moet zo eenvoudig mogelijk, maar niet eenvoudiger'. «



'Alles moet zo eenvoudig mogelijk, maar niet eenvoudiger'.