Edwin Jongsma is Practice Director Solution Development bij Avanade

Domain Specific Language

SOFTWARE FACTORIES ONTRAADSELD

Een probleem in de software-industrie is dat programmeurs te veel vrijheid hebben bij het interpreteren van de specificaties en het schrijven van code. De mogelijkheden van programmeertaal in combinatie met het .NET Framework zijn bijna oneindig. Momenteel ontwikkelt Microsoft tools voor Visual Studio waarmee Software Factories gebouwd kunnen worden. Deze tools, genaamd Domain Specific Language (DSL) tools, zijn het onderwerp van dit artikel. Deze tools kunnen de volwassenheid van de huidige IT-industrie vergroten.

De programmeertaal COBOL stamt uit 1959. RAD-talen als Delphi en Visual Basic stammen uit het begin van de jaren '90. De methodologie om software te maken is aan verandering onderhevig. De watervalmethode in combinatie met COBOL is volwassen, maar dat mag ook wel voor een taal die stamt uit de jaren '50. Het grote nadeel van de watervalmethode echter is de snelheid en flexibiliteit waarmee kan worden ontwikkeld. Er zijn praktijkvoorbeelden van applicaties waarbij een groot team twee jaar aan een applicatie heeft gewerkt, die bij oplevering is achterhaald door de technologische ontwikkelingen: schadepost miljoenen. Waar in COBOL meestal gebruik wordt gemaakt van een watervalmethode, wordt met behulp van modernere technieken vaak gebruik gemaakt van een iteratief model. Het voordeel: flexibiliteit en snelheid. Ten opzichte van COBOL zijn we nog maar kort bezig met RAD-tools. Hierdoor kunnen we stellen dat er nog veel moet gebeuren willen we onze industrie volwassen mogen noemen.

De kern van de problematiek schuilt in het feit dat een 3GL-taal te veel vrijheid biedt aan de programmeur. De mogelijkheden van een 3GL-taal in combinatie met het .NET Framework zijn bijna oneindig. Er zijn 20 miljoen programmeurs in de wereld. Een beperkt deel van deze groep kan de oneindigheid goed overzien, de meesten niet. De gevolgen zijn onder andere:

- het wiel opnieuw uitvinden
- 'not build here'-syndroom
- stovepipe solutions
- meer dan 50% IT projecten lopen niet goed[1]
- onvoorspelbaarheid proces softwareontwikkeling (duur, verloop, kosten, enzovoort)
- onvoorspelbaarheid eindproduct (kwaliteit, functionaliteit, enzovoort)

Er zal wat moeten gebeuren om deze problemen structureel op te lossen. Dit vraagt om een paradigmashift die verder gaat dan objectoriëntatie, minder programmeren en meer configureren. Een stap waar onze MSCE-vrienden al veel verder mee zijn.

Het probleem is dat er geen vastomlijnde en geïntegreerde methode is om van concept/ontwerp/model naar product te komen. Huidige programmeertalen faciliteren een dergelijke methode niet, met alle problemen van dien. De watervalmethode in combinatie met COBOL kende deze problemen minder. De programmabouwer kreeg een technisch ontwerp in handen dat weinig ruimte liet voor interpretatie. Hetzelfde geldt voor de overgang tussen functioneel en technisch ontwerp. De beste programmeurs werden functioneel ontwerper. Een DSL zorgt er ook voor dat de programmeur minder vrijheid krijgt. Een DSL genereert 3GL-code op basis van een model. De beste programmeurs maken de DSL. Dit model verhoogt het abstractieniveau voor programmeurs die ermee werken.

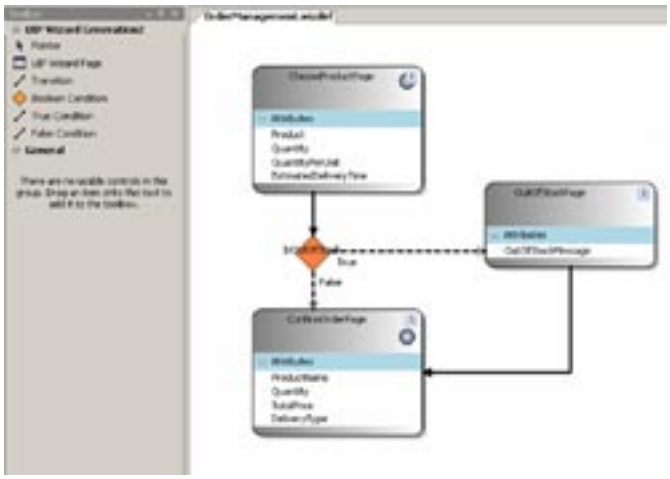
Analogie

Als je aan een Software Factory denkt moet je niet denken aan een fabriek uit de 19de eeuw, denk eerder aan een fabriek waar BMW's worden gemaakt. De BMW 3-serie is een domein binnen de fabriek. Het prototype voor de nieuwe 3-serie is gemaakt door een afdeling van de beste engineers. De investering om te komen tot het beste prototype is aanzienlijk, maar als dit eenmaal gereed is kan de lopende band worden aangezet. Er zijn meer modellen van de 3-serie die allemaal hetzelfde chassis delen, maar bijvoorbeeld de motoren zijn verschillend. Hiermee komt BMW toch tegemoet aan verschillende wensen bij de klant, terwijl 80% van de modellen gebruikmaken van dezelfde onderdelen.

Wat component-based development had moeten doen om dit voor elkaar te krijgen in onze industrie gaan DSL's hopelijk realiseren. Het prototype voor de 3-serie is vergelijkbaar met een DSL. Het specifieke domein is de auto-industrie, nog specifieker een BMW 3-serie. De BMW-engineers maken gebruik van CAD/CAM-modellen om de auto te ontwerpen. Vergelijk een DSL met deze 3D-modellen. De beste IT-architecten ontwikkelen dit model. De overige programmeurs passen deze DSL's toe om er de verschillende eindproducten voor klanten mee te realiseren. Als het goed is zijn de meeste zaken configureerbaar: kleur, type motor, interieur. Er moet ruimte zijn om handmatig een uitbreiding te doen, bijvoorbeeld vervanging van schokdempers en velgen. Om DSL succesvol te laten zijn, moet er voldoende ruimte blijven voor de programmeur om uitbreidingen te doen, maar het zou fijn zijn als deze vrijheid beperkt blijft tot 20% van het eindproduct.

Business

De grote les die onze opdrachtgevers hiervan moeten leren is dat het maken van een DSL een grotere investering vergt dan het maken van een gemiddeld softwareprogramma. De vraag is wie deze verantwoordelijkheid gaat nemen? Een opdrachtgever bij een verzekeraar bijvoorbeeld ziet misschien het voordeel niet om een softwareprogramma zo te ontwikkelen dat het vaker verkocht kan worden, omdat het gemaakt wordt voor één specifieke situatie. Een system integrator komt echter over de vloer bij verscheidene verzekeraars. Er zijn ook legio voor-



Afbeelding 1. Bouw je eigen DSL

Afbeelding 2. De gegenereerde productpagina

beelden denkbaar waar een interne IT-afdeling vergelijkbare producten levert aan meer businessafdelingen in zijn bedrijf. Zij zouden deze visie wel moeten hebben.

Voorbeelden

Een veelgebruikt voorbeeld van een DSL is de UI-designer van Visual Basic; de omgeving die de ontwikkeling van Windows-applicaties revolutionaliseerde aan het begin van de jaren '90. De BASIC-taal is 3GL, maar de UI-designer wordt vaak beschouwd als een 4GL-uitbreiding. De UI-designer is een model dat 3GL-code genereert. Het model biedt een abstractieniveau voor programmeurs die daardoor minder fouten kunnen maken en productiever zijn. Een ander voorbeeld is de orchestration designer van BizTalk. Goed om te weten dat de ontwikkeling die ten grondslag ligt aan deze designer hebben geleid tot de DST-tools. De Windows Workflow Foundation grafische designertool is een goed voorbeeld van een DSL; deze zijn deels gemaakt met behulp van de DSL-tools. Microsoft levert een aantal DSL-tools mee met Visual Studio Team System: class designer, application connection designer, logical datacenter designer en deployment designer. Een DSL hoeft geen grafisch model te zijn zoals de UI-designer. SQL is een ander voorbeeld van een DSL.

Met behulp van de DSL-tools van Microsoft kun je je eigen DSL bouwen. Hier volgt een voorbeeld van een DSL gebouwd met behulp van de DSL-tools van Microsoft. Het betreft een wizard-applicatie voor het bestellen van een product. Afbeelding 1 toont het model met welke deze wizard geconfigureerd kan worden. Lees waarmee de velden op de formulieren kunnen worden ingesteld, evenals de navigatie tussen de formulieren.

Let wel, dit model is gegenereerde code op basis van de DSL-designer. De DSL-designer is het model van het model. Je bouwt dus je eigen DSL met behulp van een DSL van Microsoft.

Op basis van dit model kun je applicatiecode laten genereren. Het resultaat van de productpagina ziet er als volgt uit; zie hiervoor afbeelding 2. De gegenereerde code heeft een 3-lagen architectuur opgeleverd, waarbij caching is toegepast om ervoor te zorgen dat de dropdown-informatie op de formulieren uit de database snel op het scherm komt.

Theorie

Voordat we het over de DSL's gaan hebben, de robotarmen van de lopende band in een softwarefabriek, besteden we aandacht aan schema's en templates.

Een Software Factory Schema is het hart van een softwarefabriek. Een schema zegt wat over de aandachtspunten van de fabriek. Het aandachtspunt van een BMW-fabriek is de auto. De requirements van een auto zijn dat je ermee van A naar B moet kunnen bewegen. In het geval van BMW is de sportiviteit belangrijk. Het schema is een overzicht van alle 'viewpoints' die relevant zijn voor de fabriek: requirements, uitdagingen. Deze requirements kunnen zowel conceptueel zijn als fysiek. Vergelijk het tot standkomen van een Software Factory Schema met het maken van een ontwerp voor een applicatie, alleen nu probeer je geen stovepipe te ontwerpen, maar een generiek framework waarmee een familie van producten kan worden gegenereerd. Het proces om tot een schema te komen is essentieel, de scope van dit artikel laat echter niet toe daar dieper op in te gaan.

Een Software Factory Template is een verzameling van assets die gebruikt kan worden om de uitdagingen zoals gedefinieerd in een schema te lijf te gaan. Een template is de implementatie van een schema, zij bevat framework-applicaties, design patterns, componenten, sourcecode, voorbeeldcode. Vaak hebben interne IT-afdelingen en system integrators wel bestaande componenten die kunnen worden hergebruikt. Een goed voorbeeld is EntLib, Avanade heeft dit in eerste instantie voor zichzelf ontwikkeld onder de naam ACA.NET. Het zal de lezer niet verbazen dat Avanade bezig is om ACA – een superset van EntLib – om te bouwen naar een Software Factory. (Zie ook het artikel van Dennis Mulder over EntLib in .NET Magazine #7)

Een Software Factory hoeft geen DSL te zijn, maar meestal bevat een Software Factory meer DSL's. Stel jezelf de vraag: kan ik een zogenaamd 'schema viewpoint' automatiseren? Als het antwoord op deze vraag positief is, heb je een potentiële DSL gevonden. Niet alle schema viewpoints kun je oplossen met automatisering. Nu denk je wellicht: wat is dan het verschil tussen hergebruik van een component, source-code zoals dat nu meestal gebeurt en een DSL? Een DSL biedt een abstractielaag waarbinnen de objecten geconfigureerd kunnen worden, zoals de UI-designer van .NET. DSL's gaan verder dan hergebruik van objecten. Vaak zie je dat organisaties best in staat zijn een goed applicatie-framework te bouwen dat bestaat uit een aantal herbruikbare componenten. Een framework bouwen dat je op Visio-achtige wijze kunt configureren - waarbij codegeneratie wordt toegepast - is echter een stuk complexer. Microsoft neemt deze complexiteit weg door middel van de DSL-tools.

DSL van de DSL

De laatste versie van de DSL Tools[2] die Microsoft beschikbaar stelt tijdens het schrijven van dit artikel is de release van 16 september. De verwachting is dat Microsoft begin volgend jaar versie 1 uitbrengt. De DSL Tools bieden een DSL om DSL's mee te maken. Het maken van een DSL is niet zo intuïtief en eenvoudig als je van Microsoft zou verwachten. Het doet meer denken aan C++ met MFC dan aan Visal Basic .NET. Niet iedereen zal echter een DSL gaan maken, het gaat erom dat mensen de gemaakte DSL gaan

Afbeelding 3. Omgeving waar de code gegenereerd wordt.

hergebruiken. Dat is een stuk eenvoudiger. De belangrijkste praktische voordelen die Microsoft met de DSL Tools voor ogen heeft zijn:

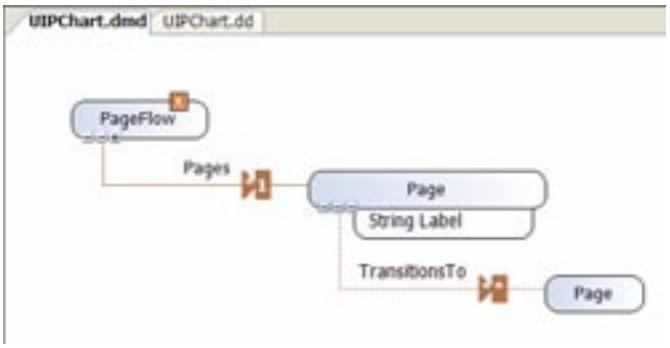
1. Een tekenomgeving; hiermee kun je eenvoudig zelf bepalen hoe je jouw DSL er grafisch uit wilt laten zien, zie afbeelding 1.
2. Een metadefinitie; de door jou gedefinieerde metadefinitie is als API beschikbaar in de codegeneratieomgeving; zie afbeelding 3 voor de genereerde HTML-code. Helaas heb je nog geen intellisense in het codegeneratie-framework.
3. Een synchronisatie-framework; aanpassingen in het model en in de gegenereerde code lopen synchroon. Goed voorbeeld hiervan is de application connection designer van Team System. Eerlijk gezegd heb ik dit nog niet aangetroffen in de DSL Tools.

Voor een voorbeeld van een DSL van een DSL, zie afbeelding 4. Dit is een model dat de navigatie van een wizard-applicatie weerspiegelt. Met dit model kun je een DSL bouwen zoals weergegeven in de paragraaf 'Voorbeelden'. De DMD-file is een DSL domain model, afgekort DMD. Naast de DMD is de volgende file belangrijk, de DSL notation definition, afgekort DD. Zie afbeelding 5.

De DMD- en de DD-file vormen beide de basis van de DSL van de DSL. In de DMD-file leg je grafisch de structuur vast, deze komt in de DD-file terug. In de DD-file kun je aangeven hoe het tekenblad van de te genereren DSL eruit gaat zien, of het plaatje dat een pagina weergeeft bijvoorbeeld blauw of rood is.

Tips & tricks

Je zou verwachten dat wijzigingen in de DMD-file direct worden weerspiegeld in de DD-file, niets is minder waar. Het DMD-model dat je bedenkt op basis van je Software Factories Schema helpt om een conceptueel idee inzichtelijk te maken. Alle classes en relaties ertussen zul je met de hand in de DD-file moeten aanpassen. Er is een tool van een 3d-party: DMD -> DD. Deze tool genereert een DD op basis van de DMD. Dit werkt goed als je de DMD in een keer goed ontwerpt. Aanpassingen naderhand zijn echter geen garantie voor de toekomst. Microsoft geeft aan dat in versie 1 van de DSL Tools de koppeling nog steeds handmatig is. Aan het begin zijn deze aanpassingen lastig, aan het eind vooral vervelend. Maar je raakt er wel aan gewend en het is beter dan niets. Schrik niet van de vele foutmeldingen die de DSL-compiler geeft als het model en de definitie niet gelijk zijn. Pak de melding die betrekking heeft op de DD-file en los deze op. Als je dat doet zullen een hele reeks bijbehorende verwarrende foutmeldingen automatisch verdwijnen. Alle classes die je grafisch beschikbaar wilt hebben in je eigen DSL



Afbeelding 4. Weergave van de DMD (DSL domain model)

moet je koppelen via een 'embedded' relatie aan de zogenaamde XML-root. Dit is de root van alle classes in het domeinmodel (in ons voorbeeld pageflow). Je kunt dezelfde classes hergebruiken in het model op een lager liggend niveau met behulp van een 'reference' relatie.

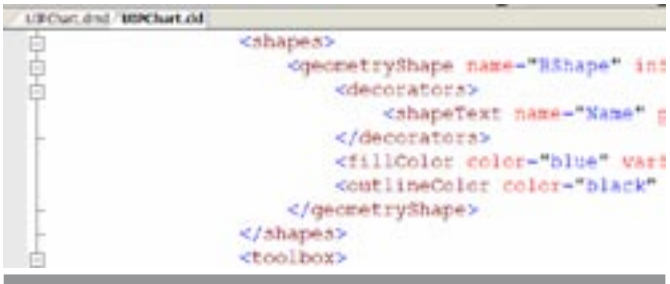
UML

De DSL Tools bevatten een aantal template-DSL's die je kunt gebruiken. De basistemplate 'minimal language' is een startpunt voor je eigen DSL. Nieuwe templates in de septemberrelease zijn: Class Diagrams, Activity Diagrams en Use Case Diagrams. Er wordt gewerkt aan Sequence Diagrams. Unified Modelling Language heeft de software-industrie veranderd, maar nog niet genoeg. In de praktijk staat UML te ver af van gegenereerde code. Boilerplate class-code genereren op basis van class diagrams is niet genoeg, het gebruikte voorbeeld in dit artikel genereert niet alleen wrappercode, maar ook formulieren met ingebouwde caching. DSL's staan veel dichter bij de programmatuur, maar je kunt er ook conceptuele ontwerptechnieken mee bouwen zoals Use Case Diagrams. Een ander goed voorbeeld in dit kader is de deployment designer van Team System. Hiermee kun je valideren of jouw applicatie op een server met bepaalde instellingen van bijvoorbeeld IIS zal draaien. Probeer dat maar eens met deployment diagrams van UML. UML staat te ver af van de realiteit waarin een programmeur bezig is en wordt daardoor in de praktijk vooral gebruikt als documentatie. Dat is goed, maar niet genoeg om ervoor te zorgen dat onze industrie volwassen wordt.

Het grote voordeel van UML is dat de verschillende modelleringtechnieken zijn samengebracht onder één noemer. In de wereld van DSL's wordt dit weer losgelaten, dat is de grote kritiek op DSL's. Microsoft gelooft niet in een generieke oplossing voor alle problemen van de hele IT-wereld. Dit verklaart ook de term domeinspecifiek. Microsoft gelooft in specifieke oplossingen voor specifieke problemen. Als het DSL-denken aanslaat, dan kunnen we uitkijken naar een toekomst van branchespecifieke generieke applicatieframeworks waarmee eenvoudiger nieuwe enterprise-schaalbare applicaties kunnen worden neergezet. Sneller en met minder bugs dan nu vaak de realiteit is.

Nog niet volwassen genoeg

De IT-industrie is lang niet zo volwassen als je deze vergelijkt met bijvoorbeeld de auto-industrie. Programmeurs hebben te veel vrij-



Afbeelding 5. Structuur van de DD (DSL notation definition)

heid in de huidige 3GL-talen en onze opdrachtgevers investeren meestal met een korte termijnvisie. Hierdoor ontstaan vaak stovepipe-oplossingen die slecht herbruikbaar zijn. Objectoriëntatie, UML en component based development hebben een goede invloed gehad, maar dat is nog niet genoeg. De DSL Tools die Microsoft biedt zijn nodig om onze industrie volwassener te maken. We staan nog aan het begin van deze paradigmashift. De tools zijn nog niet af en zullen bij versie 1 ook nog voldoende te wensen overlaten. Ik verwacht dat het nog jaren zal duren voordat de markt de voordelen ervan gaat voelen. Maar dan moeten we nu wel beginnen.

Edwin Jongsma is Practice Director Solution Development bij Avanade. Avanade is een joint venture van Accenture en Microsoft. Voor vragen en opmerkingen is hij te bereiken via edwinj@avanade.com

Nuttige internetadressen
http://lab.msdn.microsoft.com/teamsystem/workshop/sf/default.aspx
http://lab.msdn.microsoft.com/teamsystem/workshop/dsltools/default.aspx
http://forums.microsoft.com/msdn/ShowForum.aspx?ForumID=61
http://www.softwarefactories.com
http://blogs.msdn.com/jackgr/
http://blogs.msdn.com/keith_short/
http://www.modelisoft.com/Dmd2Dd.aspx

Noten

1 “The US loses 81 BILLION DOLLARS every year due to bad software practices in corporate America” (according to the Standish Group)
2 http://lab.msdn.microsoft.com/teamsystem/workshop/dsltools/default.aspx