Enterprise

Domein specifieke talen, of Domain Specific Languages (DSL) in het Engels mogen zich verheugen in een groeiende populariteit. Microsoft heeft binnen zijn software factories initiatief de DSL Tools ontwikkeld. Met dit het gereedschap is het mogelijk om eenvoudig zelf domain specifieke talen te definiëren en vervolgens te gebruiken. Bij het definiëren van een DSL dienen drie aspecten ontwikkeld te worden.

Het ontwikkelen van domein specifieke talen

Een stappenplan

Jos Warmer
Partner Ordina

Als eerste dienen de *concepten* van de taal gedefinieerd te worden. We identificeren hiervoor de concepten die in het domein een rol spelen en bepalen van deze concepten de eigenschappen en de onderlinge samenhang. De concepten worden beschreven door middel van een model. Binnen de DSL Tools heeft dit het *domein model*, daarbuiten meestal het *metamodel*, ook wordt hiervoor de term *abstracte syntax* gebruikt.

Vervolgens dienen we na te denken over het uiterlijk van de DSL. Hoe worden de concepten concreet zichtbaar gemaakt. Hiervoor worden bij een visuele taal vaak standaard vormen als rechthoeken, cirkels, ovalen, verbindingslijnen, en nog veel meer gebruikt. De DSL Tools noemt dit de *notatie* of *presentatie*. Dit aspect van een DSL wordt ook wel de *concrete syntax* genoemd.
De derde component van een DSL is de codegeneratie. De *codegeneratie* bepaalt hoe een DSL model omgezet wordt naar onderliggende code. Code is hier een breed begrip, dit omvat onder meer C# code, Java code, XML documenten, configuratiebestanden, HTML, APS, JSP, etc. etc. . Binnen de DSL Tools worden hiervoor T4 templates gebruikt. Codegeneratie is een manier om de *semantiek* of de *betekenis* van een DSL te beschrijven. Hiervoor worden ook wiskundige methodes gebruikt, maar in het kader van software ontwikkeling is codegeneratie het meest praktisch.

Als alle aspecten van een DSL beschreven zijn, dan kan er met behulp van de DSL Tools van de ontwikkelde DSL een zogenaamde setup gemaakt worden. Deze *setup* wordt vervolgens bij ontwikkelaars geïnstalleerd in Visual Studio. Hierbij krijgt de ontwikkelaar een editor (designer) ter beschikking om DSL modellen te maken welke volledig geïntegreerd is in Visual Studio. te wijzigen. De ontwikkelaar kan vervolgens DSL modellen maken, en hierbij de code genereren. Het hele proces wordt getoond in figuur 1.

In dit artikel zullen de Microsoft DSL Tools als voorbeeld gebruikt worden, maar de werkwijze en concepten zijn onafhankelijk van de specifieke tools. Ook in de Java wereld zijn tools om DSL's te definiëren ter beschikking. Binnen het Eclipse project bestaat daarvoor het Eclipse Modeling Framework (EMF) voor het definiëren van de abstracte syntax en het Graphical Modeling Framework (GMF) voor het definiëren van de concrete syntax. Voor het genereren van code bestaan meerdere mogelijkheden. De Java Emitter

Templates (JET) is een tegenhanger van de T4 templates. De Xpand2 taal van openArchitectureWare is een template taal met geavanceerdere mogelijkheden. Een ander aanpak is om gebruik te maken van gespecialiseerde DSL tools. Een goed voorbeeld daarvan is MetaEdit, een standalone applicatie waarmee DSL's gemaakt kunnen worden.

1. Hoe ontwikkel je een DSL?

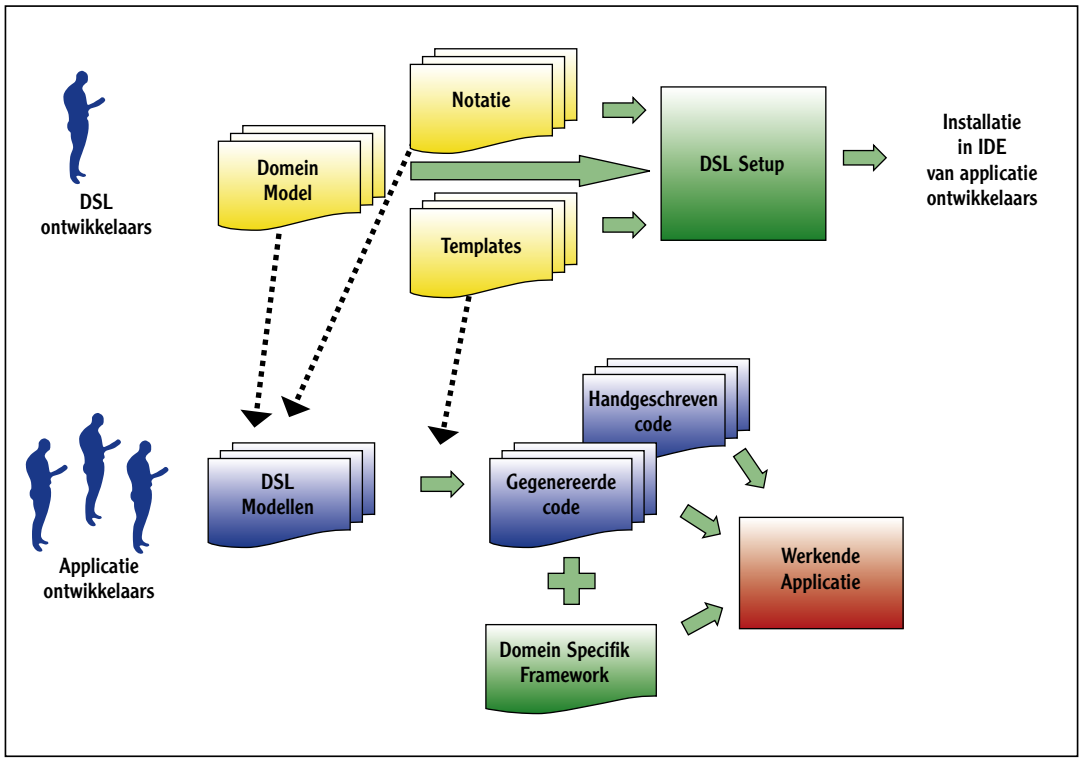
Wanneer iemand begint met de DSL Tools is het geheel natuurlijk om de werkwijze uit figuur 1 letterlijk te volgen. Je begint zo snel mogelijk met het domain model en de notatie te definiëren en je hebt al snel een mooie visuele editor. Vervolgens denk je na over de te genereren code en schijf je wat templates. Alle walkthroughs en tutorials werken op deze wijze. Deze aanpak is uitstekend voor exploratie, wanneer je bezig bent om de DSL Tools te ontdekken en leert ze te gebruiken.

Als je de exploratiefase achter je laat en een DSL wilt ontwikkelen die daadwerkelijk gebruikt gaat worden werkt bovenstaande aanpak averechts. Zo een aanpak lijkt een beetje op zomaar beginnen met programmeren zonder na te denken over de requirements of de gewenste functionaliteit.

Als je een DSL gaat ontwikkelen is het belangrijk om eerst na te denken over het doel van de DSL, en het resultaat dat je wilt bereiken met het gebruik van de DSL. In dit artikel beschrijven we de werkwijze die we gevolgd hebben bij het ontwikkelen van de vier DSL's in de SMART-Microsoft Software Factory. De beschreven werkwijze is grotendeels onafhankelijk van de gebruikte DSL Tools, en ook toepasbaar in bijvoorbeeld de Eclipse omgeving.
De context van SMART-Microsoft is het ontwikkelen van DSL's waarmee ontwikkelaars snel applicaties kunnen maken die voldoen aan alle architectuurrichtlijnen. De DSL's dienen met name ook bruikbaar te zijn voor medior ontwikkelaars.

1.1 Stap 1 – definieer de referentie-architectuur

De eerste stap is het selecteren van het doelplatform. Ook als we uiteindelijk een platformonafhankelijke DSL willen maken is deze stap van belang, omdat de codegenerator een onderdeel van de DSL definitie is. De selectie van het doelplatform vindt op verschillende detailniveaus plaats. In eerste instantie kiezen we voor de gewenste architectuur, wat een beschrijving van het doelplatform op hoog niveau oplevert. Omdat we werkende code willen genereren is een high-level architectuur niet voldoende. Voor alle onderdelen in de architectuur maken we vervolgens concrete keuzes voor de te gebruiken componenten en frameworks. Op deze manier vullen we de architectuur concreet in zodat we met de gekozen componenten direct kunnen beginne n met de bouw.



Figuur 1

Ga er vanuit dat je meestal meerdere DSL's nodig hebt om het probleem op te lossen

1.2 Stap 2 – bouw een referentieapplicatie

Voor het schrijven van een codegenerator geldt natuurlijk dezelfde regel als voor de DSL als geheel, direct een codegenerator maken zal niet werken. Je kunt geen codegenerator schrijven als je niet eerst in staat bent om de code met de hand te schrijven. Daarom is de tweede stap het met de hand bouwen van een referentieapplicatie volgens de doelarchitectuur die gebruik maakt van alle geselecteerde componenten en frameworks.

Deze stap is feitelijk een klein software project. Je stelt functionele specificaties op, al dan niet in de vorm van use cases of scenarios. Vervolgens maak je een ontwerp en bouw en test je de applicatie.

In deze stap valideer je alle keuzes uit stap 1. Werken de gekozen frameworks wel samen, bieden ze genoeg functionaliteit, zijn ze wel compleet genoeg, enzovoorts. Als gevolg hiervan kunnen keuzes uit stap 1 heroverwogen worden. Ook komt het voor dat hier stukjes eigen framework ontwikkeld worden als dat nodig blijkt. Omdat deze referentieapplicatie de basis is voor het ontwikkelen van de DSL is het belangrijk om de code uitgebreid te reviewen door een team van ervaren architecten en ontwikkelaars.

Het kan voorkomen dat er al een of meer referentieapplicaties aanwezig zijn. In dit geval kan en bestaande applicatie gebruikt worden. Wel dient deze gereviewd te worden tegenover alle keuzes uit stap 1. Waarschijnlijk zullen bestaande applicaties ook enige refactoring nodig hebben om op alle punten geheel conform de referentiearchitectuur te zijn.

1.3 Stap 3 – analyseer de code

In stap 3 gaan we de referentieapplicatie uit stap 2 analyseren. Deze code van deze referentieapplicatie valt uiteen in drie delen, welke in het vervolg verschillend behandeld zullen worden.

1.3.1 Statische code

Het eerste deel is de code die identiek is voor iedere applicatie. Deze code wordt straks omgezet naar herbruikbare bibliotheken, componenten of frameworks.

1.3.2 Unieke code

Het tweede deel van de code is de code die per applicatie uniek is. Dat wil zeggen dat de code geheel afhankelijk is van de specifieke applicatie. Het is niet zinvol deze code te genereren omdat zo een generator slechts binnen één applicatie gebruikt kan worden. Deze code zal dan ook met de hand geschreven blijven worden en geen impact hebben op de DSL definitie.

1.3.3 Terugkerende patronen

Het derde deel van de code is die code die in het type applicaties dat we willen ontwikkelen gelijksoortig is. Dat wil zeggen dat de code wel hetzelfde patroon volgt, maar dat de invulling van het patroon afhankelijk is van de specifieke applicatie. Dit soort code leent zich specifiek om met behulp van DSL modellen te genereren. De patroonmatige code vormt straks de basis voor de codegenerator. De codegenerator zal gestuurd worden door de DSL modellen.

1.4 Stap 4 – vind de DSL concepten

In stap vier gaan we op zoek naar de concepten zoals die in het DSL domein model dienen te komen. Deze concepten worden afgeleid uit de geanalyseerde code uit stap 3. We zoeken de patronen in de te genereren code en definiëren op basis daarvan de concepten in de DSL. Het is hier belangrijk om de focus te leggen op concepten die op een hoger abstractieniveau liggen dan de programmeertaal. Verder dient het mogelijk te zijn om uit de concepten de benodigde code te genereren.

Een vuistregel die we gebruiken is dat een DSL niet teveel concepten moet bevatten. Ieder concept in een DSL wordt meestal een icoontje op een toolbar. De vuistregel is dat alle icoontjes altijd zichtbaar moeten zijn, dus het aantal concepten dient beperkt te worde tot maximaal een stuk of vijftien. Deze vuistregel is sterk ingegeven door onze doelstelling dat een DSL eenvoudig te gebruiken dient te zijn. Als een DSL net zo complex wordt als de onderliggende programmeertaal missen we het doel.

Ga er vanuit dat je in een gegeven situatie meestal meerdere DSL's nodig hebt om het probleem op te lossen. Een DSL is klein, en dat moet je zo houden, probeer vooral niet om zoveel mogelijk in één DSL te proppen. Dat betekent wel dat er voor serieuze problemen al snel meerdere DSL's nodig zijn. Probeer de verschillende domeinen van deze DSL's zo vroeg mogelijk te bepalen. In SMART-Microsoft hebben we bijvoorbeeld vier DSL's ontwikkeld die ieder een bepaald deel van de architectuur afdekken. Hierdoor zijn de DSL's klein en eenvoudig gebleven,

Op basis van de gevonden concepten bepalen we vervolgens de notatie. Hierbij hebben we gebruik gemaakt van storyboarding. Dat wil zeggen dat we DSL modellen eerst in Visio getekend hebben als voorbeeld. Het is gebleken dat dit een vruchtbare aanpak is. Door een DSL model daadwerkelijk uit te tekenen leer je zo snel mogelijk waar problemen gaan optreden. Vaak blijkt dat een probleem om de juiste notatie te vinden voorkomt uit een verkeerde keuze van de concepten. Hier ontstaat

een aantal korte iteraties waarin de concepten telkens duidelijker worden.

1.5 Stap 5 – maak het domein model en de notatie

In deze stap maken we het domein model voor de DSL op basis van de resultaten uit stap 4. Dat wil zeggen dat van alle concepten de eigenschappen en de samenhang met de andere concepten bepaald dient te worden. Dit domein model wordt in de DSL Tools gemaakt. Vervolgens gaan we de notatie uit de storyboards ook in de DSL Tools inbrengen.
Een belangrijke regel die we gebruiken is dat alles in een model gebruikt moet worden om iets te genereren. We modelleren niet als documentatie. Zowel de concepten, de bijbehorende eigenschappen, alsook de relaties tussen de concepten dienen allemaal bij te dragen aan een stukje codegeneratie. Dit aspect wordt blijvend gevalideerd.

1.6 Stap 6 – code generatie

Nadat het domein model en de notatie vastgelegd zijn gaan we werken aan de codegeneratie. Voor alle concepten bepalen we welke code er gegenereerd dient te worden. De code uit de referentieapplicatie dient hierbij als voorbeeld. Ten slotte willen we die code kunnen genereren. In de code worden de variabelen van het patroon gevonden en vervangen door parameters die vanuit een DSL model gevuld worden. Zo wordt de code geparametriseerd totdat het een nette template is geworden.
In deze stap worden lastige keuzes gemaakt. Zo kan voor het genereren van code gekozen worden voor het maken van een vast framework en het genereren van code die op het framework werkt. Dat betekent minder code die gegenereerd wordt. Als alternatief kan er volledige code gegenereerd worden zonder een framework te ontwikkelen.

In het geval; van een handgeschreven applicatie gaat de voorkeur meestal uit naar het maken van zoveel mogelijk frameworks. Die worden hergebruikt, en de hoeveelheid handgeschreven code wordt geminimaliseerd. Als we de code gaan genereren kunnen hele andere afwegingen gemaakt worden. Een framework is vaak complex en lastig te doorgronden, wat pleit voor meer gegenereerde code en minder frameworks. Daarmee wordt de gegenereerde code wellicht leesbaarder. Ook is het ontwikkelen van een framework complex en tijdrovend en kan het veel gunstiger zijn om de equivalente code direct uit te genereren.

Een aspect dat veel aandacht vergt is het hergenereren van code uit DSL modellen. Een noodzakelijke randvoorwaarde voor het succesvol gebruik maken van DSL's is dat het DSL model altijd leidend blijft. Dat wil zeggen dat er te allen tijde opnieuw code gegenereerd kan worden, zonder de handgeschreven delen van de code te overschrijven. Deze eis stelt sterke voorwaarden aan de structuur van de gegenereerde code. Zo is het goed om in dit stadium na te denken op welke wijze en op welke plaats handgeschreven code toegevoegd dient te worden. De plaatsen waar dit mogelijk is worden ook wel *extension points* genoemd, Naast het ontwerpen van deze extension points dienen ze ook gedocumenteerd te worden. Alle ontwikkelaars die de DSL's gaan gebruiken dienen deze extension points goed te kennen zodat ze weten waar met de hand code toe te voegen.

Omdat de keuzes in deze stap lastig zijn worden ze vaak iteratief bepaald tijdens het ontwikkelen van de codegenerator.

1.7 Stap 7 – herbouw de referentieapplicatie

Als stap 6 afgerond is kunnen we de referentieapplicatie herbouwen met gebruik van de DSL. Deze referentieapplicatie geldt als d test van de DSL. Kunnen inderdaad bouwen wat we wilden? En is de gegenereerde applicatie correct en zonder fouten? Zo ja, dan zijn we klaar voor productie.

1.8 Stap 8 – creëer een vruchtbare omgeving voor het gebruik

Een succesvolle DSL ontwikkelen is niet klaar bij stap 7. Het succes van een DSL ligt hem immers in het gebruik ervan, niet in de definitie op zichzelf. Het is daarom belangrijk om energie te steken in het ondersteunen van het gebruik van de DSL's.

Ten eerste dient er trainingsmateriaal ontwikkeld te worden om de ontwikkelaars een workhop o.i.d. aan te kunnen bieden waarin ze leren om te gaan met de ontwikkelde DSL's. Enige kennis en uitleg van de onderliggende architectuur en componenten is hierbij ook nodig. De referentieapplicatie kan ook gebruikt worden bij het ontwikkelen van trainingmateriaal zoals en workshop. De deelnemers hebben direct een volledige voorbeeldapplicatie ter beschikking.

Een tweede aspect is de ondersteuning van projecten die de DSL's gebruiken. Als de DSL ontwikkelaar zijn eigen DSL gebruikt zal het wel goed gaan, maar het gaat erom dat een grote groep ontwikkelaars met de DSL aan de gang kan. Het is goed om het eerste project uit te voeren met ontwikkelaars die niets met de ontwikkeling van de DSL's te maken hebben gehad. Dat geeft een goed beeld over de bruikbaarheid van de DSL's door de

doelgroep. Wel is het handig om minimaal één van de DSL ontwikkelaars in een ondersteuningsrol aan het project te koppelen. Zeker in een eerste project kom je tal van kleine kinderziektes tegen en die dienen snel opgelost te worden. Tijdens het eerste project hebben we ook een Wiki opgezet waarin de vragen van de ontwikkelaars en de antwoorden van de ondersteuners neergezet zijn. Dat levert aan het eind van het project waardevolle informatie waar volgende projecten uit kunnen putten.

Een laatste aspect dat ingepland dient te worden is het release management. De DSL dient up to date gehouden te worden, uitgebreid met nieuwe en nuttige concepten, etc. het uitbrengen van nieuwe versies van een DSL dient goed georganiseerd te zijn, anders ontstaat vanzelf de traditionele versie problematiek die we op tal van plaatsen inde IT zo goed kennen.

1.9 Alle stappen - werk iteratief

Zoals altijd in een geschreven stuk lijkt de nadruk te liggen op het sequentieel uitvoeren van alle stappen. In de praktijk hebben we gezien dat de stappen 4, 5 en 6 ook vaak in een aantal; kleine iteraties worden uitgevoerd. Zodra in stap 4 een concept onderkend is, wordt een voorbeeld notatie gemaakt en wordt bekeken welke code er uit dit concept gegenereerd kan worden. Ook in stap 1 en 2 kan iteratief gewerkt worden. Bij het ontwijkkelen van meerdere DSL's kan is het zinvol om per DSL de stappen 4 tot en met 7 door te lopen, je kunt er zelfs voor kiezen om per DSL in productie te gaan, zodat de kosten zich sneller terugbetalen.

1.10 Alle stappen - Zorg voor het juiste team

Een veel gemaakte fout is het ontwikkelen van en DSL door een of twee personen in isolatie, de zogenaamde DSL specialisten. Zoals uit de werkwijze blijkt is er bij het ontwikkelen van DSL's een breed scala aan expertises nodig. Het is zaak om deze op de juiste plaats en het juiste moment in te zetten. De benodigde expertises omvatten architecten en ervaren ontwikkelaars bij het bepalen van de architectuur en de gebruikte componenten en frameworks, ervaren modelleurs en metamodelleurs voor het definiëren van de juiste concepten in een DSL, mensen met ervaring in codegeneratie, mensen met didactische vaardigheden.

Gezien de breedte van het gebied dat bij de ontwikkeling van DSL's betrokken is, is een duidelijke visie op het geheel en de samenhang van alle onderdelen van belang. Voor dit soort expertise wordt de term meta-architect wel gebruikt. Gedurende het hele ontwikkeltraject dient iemand deze rol te vervullen zodat uiteindelijk een samenhangend geheel opgeleverd ka n worden.

2 Conclusie

Het ontwikkelen van DSL's is een relatief nieuwe tak van sport in de software ontwikkeling. In dit artikel is een aanpak beschreven die helpt om vanuit het te behalen doel geredeneerd op structurele wijze te komen tot DSL's die succesvol gebruikt kunnen worden. Kijkende naar figuur 1 beginnen we feitelijk aan de achterkant bij de werkende applicatie (stap 1, 2 en 3) en gaan we daar vanuit pas beginnen aan het daadwerkelijk ontwikkelen van de DSL (stap 4, 5, 6 en 7). De SMART-Microsoft DSL's zijn inmiddels in een groot aantal projecten gebruikt en we hebben ondervonden dat de aanpak zijn vruchten heeft afgeworpen. We hebben dus veel baat gehad bij deze aanpak en denken dat deze breed toepasbaar is. Doe er je voordeel mee.

Een veel gemaakte fout is het ontwikkelen van en DSL door een of twee personen in isolatie