In de software-industrie zijn budget- en deadline-overschrijdingen aan de orde van de dag, er wordt vaak niet aan de gestelde verwachtingen voldaan. Dit kan worden voorkomen door software-ontwikkeling te automatiseren, maar dit doen we nog te weinig. Software Factories is een aanpak waarmee deze problemen kunnen worden voorkomen. In dit artikel bekijken we eerst de huidige zwaktes en hoe deze opgelost kunnen worden met de Software Factories aanpak. Daarna zullen we een belangrijk concept van deze aanpak toelichten: Domain Specific Languages of kortweg DSL's. Ten slotte laten we aan de hand van een voorbeeld zien hoe een DSL gemaakt kan worden.

Software Factories

Toepassing van Domain Specific Languages

Een van de huidige zwaktes van software-ontwikkeling is dat de 3GL-talen zoals Java of C# ons teveel vrijheid bieden. Met fundamentele zaken als loops, functies, strings en integers bouwen we complete bedrijfskritische applicaties. Om het abstractieniveau te verhogen maken we meestal modellen van applicaties voordat we deze gaan bouwen, maar als we daadwerkelijk hiermee beginnen worden deze vaak niet meer gebruikt. Resultaat: het model komt niet meer overeen met de implementatie en heeft alleen tijdelijk gefungeerd als een ontwerp- en communicatiemiddel.

Een andere zwakte is het hergebruik door middel van componenten. Deze aanpak had veel problemen moeten oplossen, maar heeft toch niet volledig gebracht wat we hadden gehoopt. We hebben nu herbruikbare frameworks met componenten, maar bij het gebruik ervan is veel onderliggende kennis vereist en ziet men bepaalde simpele handelingen regelmatig terug. We hebben nog steeds veel vrijheid hoe implementaties met frameworks worden verricht. Als we dat vergelijken met hoe we een huis bouwen dan betekent dit dat we tegenwoordig met allerlei onderdelen aankomen zoals balken, stenen, ramen, verf en bedrading. Maar die gaan we telkens weer op een unieke manier met elkaar verbinden.

De aanpak Software Factories is een aanpak om de genoemde problemen te voorkomen. Een Software Factory is een op maat gemaakte ontwikkelingomgeving die alle benodigdheden (patterns, modellen, frameworks, tools en ontwikkelprocessen) bevat om een familie applicaties te bouwen met een gemeenschappelijke architectuur. Hierbij kan gewoon gebruik worden gemaakt van bestaande technieken zoals bijvoorbeeld componenten, frameworks, aspect oriented programming en webservices. Essentieel voor een Software Factory is het gebruik van modellen en talen die domeinspecifiek zijn en in relatie met elkaar staan: de zogenoemde Domain Specific Languages. Deze talen verhogen het abstractieniveau waarop we werken en de daadwerkelijke implementatie in een 3GL-taal zal uit deze modellen worden gegenereerd. Als we dit weer vergelijken met het bouwen van een huis dan gebruiken we bij de Software Factory aanpak voorgefabriceerde muren, daken en een standaard-blauwdruk zodat we deze onderdelen vlot aan elkaar kunnen koppelen.

Eenvoudige programmeertaken zullen geautomatiseerd zijn in de factory. Dit zal leiden tot meer assembleren en configureren en tot minder programmeren. Een typische factory moet ongeveer 40% tot 80% van de implementatie kunnen automatiseren. Bij minder dan 40% is het op dit moment niet de moeite waard om een factory te bouwen, de investering zal waarschijnlijk niet worden terugverdiend. Bij meer dan 80% wordt de factory al snel te domeinspecifiek en zal dus te weinig ingezet kunnen worden om de investering waard te zijn.

Het gedeelte van de software dat uit de factory komt, heeft normaal gesproken een hogere kwaliteit dan handgeschreven code. Een Software Factory zal namelijk door de meest ervaren ontwikkelaars en architecten binnen het bedrijf worden gebouwd welke diepgaande kennis van het domein bezitten waarvoor de factory ontwikkeld wordt. De code die gegenereerd wordt, zal

Adv Microsoft

Adv Microsoft

minder fouten bevatten, uitgebreid getest zijn en zal voldoen aan de 'best practices' die in het domein van toepassing zijn. Het voorgaande klinkt erg ideaal, maar we zijn nog lang niet zo ver. De verwachting van de bedenkers van de Software Factory visie is dan ook dat Software Factories pas over tien tot vijftien jaar de standaard-aanpak voor software-ontwikkeling zijn.

Language workbenches Domain Specific Languages zijn essentieel voor Software Factories, ze moeten het abstractieniveau waarop we nu werken gaan verhogen. In het verleden kostte het veel moeite om eigen talen te definiëren en om de benodigde tools daaromheen te bouwen. Resultaat is dan ook dat de meeste DSL's die nu bestaan, gebouwd zijn voor brede horizontale domeinen met dus ook grote doelgroepen. Voorbeelden hiervan zijn GUI designers (uit bijvoorbeeld Delphi of Visual Basic), SQL en UML. Merk op dat in tegenstelling tot wat we vaak geneigd zijn te denken talen niet altijd tekstueel hoeven te zijn, een GUI designer is hier een mooi voorbeeld van.

Om de Software Factories-aanpak rendabel te laten zijn, zal het eenvoudiger moeten worden om zelf nieuwe talen te kunnen definiëren. Zogenaamde language workbenches bieden hier de uitkomst. Dit zijn tools waarmee we zelf relatief eenvoudig DSL's kunnen definiëren. Een language workbench kan je zien als een Software Factory om andere Software Factories mee te maken. De Microsoft-tools voor Domain Specific Languages (in het vervolg DSL Tools) zijn een voorbeeld van een dergelijke language workbench. De DSL-tools leveren een platform om grafische talen mee te definiëren. Op dit moment zijn deze tools nog in ontwikkeling, de huidige versie (November preview) bestaat uit de volgende onderdelen:

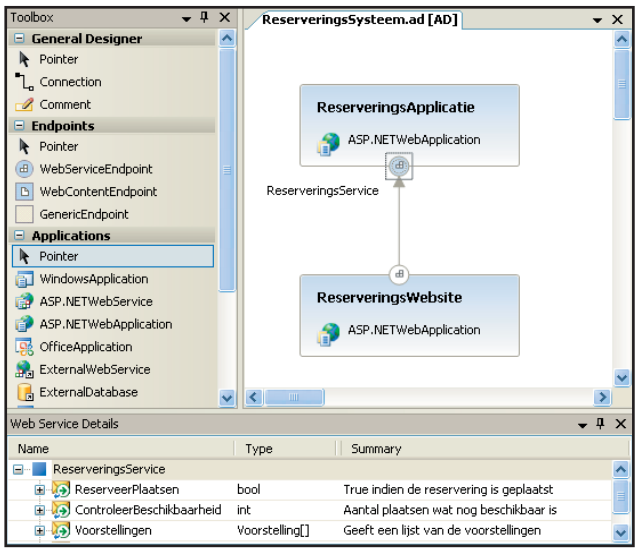
- Domain Model Framework: Met eenvoudige principes zoals classes, referenties en overerving is de taal te definiëren (zie afbeelding 3).
- Drawing Surface Framework: Dit zorgt ervoor dat de gedefinieerde taal grafisch wordt weergegeven en biedt standaard functionaliteit zoals 'drag and drop' en automatische layout.
- Validation Framework: Hiermee zijn extra regels te schrijven om te controleren of de taal correct wordt toegepast.
- Template Engine: In templates is de 3GL code te definiëren welke aan de hand van het model/taal moet worden gegenereerd.
- Shell Framework. Dit zorgt voor volledige integratie van de DSL met de ontwikkelomgeving, in dit geval Visual Studio.NET 2005.

Belangrijk is de integratie in de ontwikkelomgeving, dit zorgt er namelijk voor dat de modellen die de ontwikkelaar met behulp van de taal definieert direct kunnen worden gebruikt om gedeeltes van de software te genereren met behulp van de Template Engine. Op deze manier ontstaat het effect dat modellen belangrijker worden en ze zelfs gebruikt gaan worden om mee te programmeren.

DSL voor servicegeoriënteerde systemen

Laten we een voorbeeld geven hoe een DSL kan worden ingezet in het domein van servicegeoriënteerde systemen. Stel dat we een reserveringssysteem voor een theater willen bouwen. Bij dit systeem moet het mogelijk zijn om via een service op te vragen welke voorstellingen er zijn. Daarna kan er worden gekeken of er nog plaatsen voor een bepaalde voorstelling vrij zijn en kunnen indien gewenst plaatsen worden gereserveerd.

Microsoft heeft voor Visual Studio 2005 zelf al een DSL gemaakt voor servicegeoriënteerde systemen. Deze is voor ons als DSL-eindgebruikers beschikbaar als onderdeel van de Distributed System Designer in Visual Studio. Met deze DSL is het mogelijk verschillende applicaties in een schema te plaatsen en daaraan webservices te koppelen. In afbeelding 2 is te zien hoe ons reserveringssysteem in de Distributed System Designer er uit zou zien. De 'ReserveringsApplicatie' huisvest de service functionaliteit. In de afbeelding is het service-endpoint van de reserveringsapplicatie geselecteerd. De definitie van deze service wordt onderin het scherm getoond en bevat de eerder genoemde operaties (om het overzichtelijk te houden zijn parameters niet zichtbaar gemaakt). De 'ReserveringsWebsite' hebben we verbonden met deze service en kan zo dus gebruik maken van de geboden functionaliteit. Als ons schema klaar is kunnen we er voor kiezen om de implementatie bij dit schema te laten genereren, de projecten en code worden dan gegenereerd voor zover dit mogelijk is.



Afbeelding 2. De definitie van het reserveringssysteem in de Distributed Systems Designer.

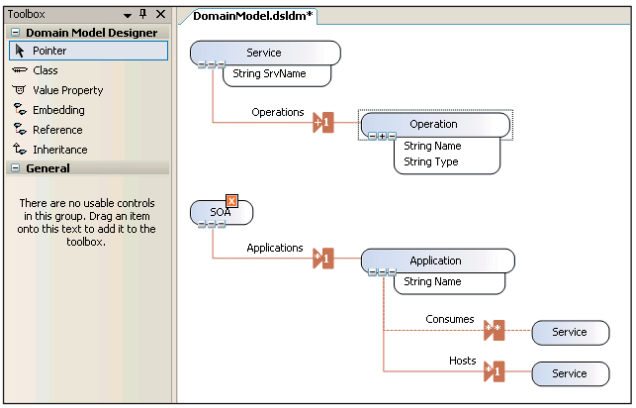
Laten we eens kijken hoe we zelf met de DSL Tools deze structuur in een DSL hadden kunnen definiëren. Dit doen we met behulp van het Domain Model Framework in een zogenaamd DSL Domain Model (DMD). Een DMD kan gezien worden als een DSL om andere DSL's mee te definiëren. Als elementen in onze servicegeoriënteerde DSL hebben we applicaties, services en operaties. In de DMD zoals weergegeven in afbeelding 3 zijn de onderlinge relaties van de elementen zichtbaar. Hierbij wordt onderscheid gemaakt in embedded relaties (doorgetrokken lijnen) waarbij child-elementen kunnen worden aangegeven en references (gestippelde lijnen) waarbij verwijzingen naar andere elementen kunnen worden gemaakt. Zo wordt in dit voorbeeld een service altijd gehost door één applicatie en geconsumeerd door een willekeurig aantal applicaties. In een volwaardige DSL zouden we natuurlijk nog meer details moeten verwerken.

DSL Notation Definition Met de DMD alleen zijn we er niet. Deze geeft namelijk niet aan hoe onze elementen gevisualiseerd dienen te worden. Dit wordt vastgelegd in de DSL Notation Definition (DD). Standaard kan een keuze worden gemaakt uit verschillende vormen en kleuren. Ook kunnen er afbeeldingen worden gebruikt en kunnen er labels worden geplaatst bij de elementen. Mochten de standaardmogelijkheden niet voldoende zijn, dan kan men altijd nog eigen visuele elementen schrijven als uitbreiding op het Drawing Surface Framework. In dit artikel gaan we niet verder in op de DD-notatie, omdat deze notatie in de definitieve versie waarschijnlijk een andere vorm zal krijgen.

Na het definiëren van de DMD en DD is de basis voor onze DSL gelegd. Als we de DSL compileren beschikken we over de mogelijkheid schema's te maken op een vergelijkbare wijze zoals in afbeelding 2 al te zien was. Hiermee wordt echter nog geen code gegenereerd. De schema's genereren al wel metadata die via een API beschikbaar wordt gesteld aan de Template Engine om code mee te gaan genereren. In ons geval zouden we templates kunnen definiëren die gebruik maken van de voordelen van het nieuwe platform van Microsoft voor servicegeoriënteerde systemen: Windows Communication Foundation (WCF). Voor de gedefinieerde applicaties kunnen we dan de servicecontracten, services en proxy's laten generen.

We hebben nu de volgende voordelen als we deze DSL gebruiken:

- De DSL genereert gedeeltes van de implementatie, hiervoor hebben we geen literatuur voor de details van de technieken door te hoeven lezen.
- Er wordt tijd bespaard omdat programmeertaken zoals het toevoegen van de juiste referenties en het markeren van methodes en klassen met de juiste attributen zijn geautomatiseerd.



Afbeelding 3. De definitie van de DSL in de DSL Tools.

- De DSL bevat elementen die domein specifiek zijn, hiermee werken we op een hoger abstractieniveau en hoeven ons minder bewust te zijn van onderliggende technische details.
- Het gebruik van de DSL is direct in de ontwikkelomgeving geïntegreerd waardoor er geen gat ontstaat tussen het model en de implementatie.

Conclusie De enorme vrijheid en vele mogelijkheden die we hebben met de huidige manier van software ontwikkelen is de oorzaak waarom veel deadlines en budgetten worden overschreden. Software Factories kunnen deze problemen voorkomen door de inzet van bestaande technieken en Domain Specific Languages. Door het toepassen van DSL's binnen de Software Factories aanpak gaat het abstractieniveau omhoog waardoor we vele malen productiever zullen zijn. Het definiëren van deze talen is met de op dit moment beschikbare tools eenvoudiger geworden. Er is nog een lange weg te gaan om ons vakgebied volwassen te maken, maar de Software Factory aanpak is veelbelovend. De eerste stappen om tot een Software Factory te komen zijn nu al te nemen en het is zeker nog niet te laat om er aan te beginnen.

Referenties

Voor meer informatie over Software Factories en DSL Tools:

- Jack Greenfield and Keith Short. Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools. Wiley, 2004.
- http://www.softwarefactories.com
- http://msdn.microsoft.com/vstudio/teamsystem/workshop/sf

Gerben van Loon en Antoine Savelkoul zijn werkzaam bij Avanade.
(E-mail: gerbenv@avanade.com of antoines@avanade.com)