Binnen Visual Studio 2005 heeft Microsoft de zogenaamde DSL Tools geïntroduceerd. Met deze tools kan iedereen zijn eigen visuele Domein Specific Language (DSL) definiëren. Hierbij kunnen zowel de taalelementen van de DSL gedefinieerd worden, alsook de visuele editor en de bijbehorende code-generatie. De resulterende DSL integreert vervolgens als een plugin geheel in VisualStudio. DSL's spelen een centrale rol binnen het Software Factory concept van Microsoft.

thema

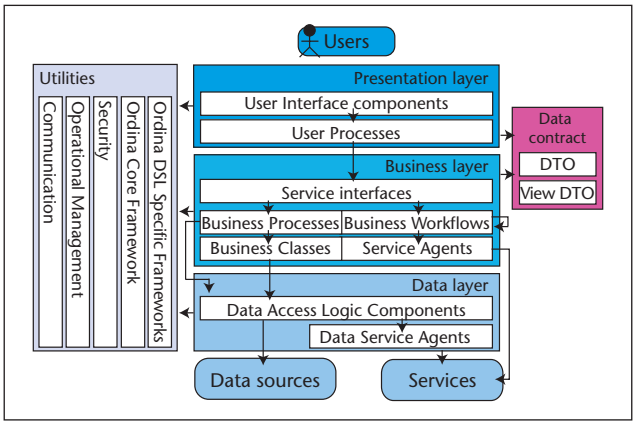
SMART-Microsoft Software Factory

Samenspel van architectuur en DSL's

In de visie van Microsoft worden DSL's ontworpen voor specifieke types applicaties, en kan met behulp van de gedefinieerde DSL's een zogenaamde productline opgezet worden.

Binnen het Development Center Microsoft van Ordina is in de eerste helft van dit jaar een software factory ontwikkeld, genaamd de SMART-Microsoft Software Factory, waarbinnen uitgebreid gebruik gemaakt wordt van DSL's. Dit artikel beschrijft op welke wijze de benodigde DSL's bepaald zijn en hoe dit heeft geleid tot een effectieve en flexibele software factory. In het eerste deel wordt beschreven voor welk type applicaties de software factory bedoeld is, hierbij speelt de architectuur van de applicaties een essentiële rol. Het tweede deel beschrijft welke DSL's we ontwikkeld hebben om dit type applicatie zo efficiënt mogelijk te kunnen ontwikkelen. Bij de ontwikkeling van de DSL's is de architectuur een van de leidende factoren geweest.

ARCHITECTUUR Moderne service georiënteerde systemen zijn steeds vaker gebaseerd op een architectuur. Door een architectuur te maken voor een systeem waarin met aspecten als onderhoudbaarheid, beveiliging, hergebruik, schaalbaarheid of beschikbaarheid uitvoerig rekening is gehouden kunnen betere systemen ontwikkeld worden. Gelukkig is het niet noodzakelijk om voor elk systeem een architectuur weer volledig opnieuw te maken. Veel *best practices* zijn al gebundeld in diverse referentie-architecturen. Microsoft heeft zo een referentie-architectuur voor het maken van systemen met .NET. De SMART-Microsoft architectuur is gebaseerd op deze Microsoft referentie-architectuur. De redenen hiervoor zijn eenvoudig. Microsoft heeft jarenlang best practices voor .NET verzameld en gebundeld in hun architectuur. Bovendien wordt deze architectuur ook goed ondersteund door standaardcomponenten en is de ontwikkelomgeving hierop afgestemd. Helaas is de referentie-architectuur van Microsoft nog niet direct bruikbaar als architectuur voor systemen waarbij code-generatie op basis van modellen wordt gebruikt. De Microsoft referentie-architectuur laat namelijk nog veel ruimte over voor de detail-invulling en de verantwoordelijkheden zijn niet altijd expliciet benoemd. Voor systemen die grotendeels op basis van modellen worden gemaakt zijn expliciete keuzes en verantwoordelijkheden echter heel belangrijk. De SMART-Microsoft archi-



FIGUUR 1. De SMART-Microsoft architectuur.

tectuur hanteert de Microsoft referentie-architectuur, zodat er een architectuur ligt waarmee op basis modellen code gegenereerd kan worden. De keuzes die gemaakt zijn in de architectuur zijn gericht op ontwikkeling van administratieve enterprise systemen.

De SMART-Microsoft architectuur is opgedeeld in meerdere lagen. Binnen elke laag zijn er verschillende onderdelen te onderkennen met specifieke verantwoordelijkheden. Van alle onderdelen volgt nu een korte beschrijving van de belangrijkste verantwoordelijkheden en hoe dat tot uiting komt bij codegeneratie vanuit modellen. In de praktijk zijn er veel meer gedetailleerde keuzes gemaakt om goede code-generatie mogelijk te maken, maar het voert te ver om die hier volledig te behandelen.

Presentatielaag De presentatielaag bestaat uit twee onderdelen: UI components en User processen. De UI components zijn webpagina's of windows forms. Deze zijn uitsluitend bedoeld om schermen te tonen en mogen dus geen logica behalve schermvalidaties bevatten. De schermen worden door User processen aangestuurd. Deze User processen zijn verantwoordelijk voor navigatie, statemanagement en het aanroepen van services. Het user proces, statemanagement en de navigatie worden ingevuld met het UI Proces application buildingblock van Microsoft. De aanroep van de services gebeurt altijd via een gegenereerde proxy.

Datacontract In een service georiënteerde architectuur zijn datacontracten een onderdeel van het service contract. Dit zijn echter tevens de data waarmee de presentatie laag moet werken. Daarom is in de SMART-Microsoft architectuur het datacontract expliciet gemaakt en niet een onderdeel van de service-interface. Datacontracten moeten voldoen aan open standaarden, in de praktijk betekent dit dat er een XSD van gemaakt moet kunnen worden. .NET specifieke types als dataset zijn dan ook niet toegestaan in het datacontract. Er zijn twee soorten datacontracten of data transfer objecten (DTO's) onderkend: data die bewerkt kunnen worden (meestal een beperkt aantal objecten) en data die in lijsten getoond worden (meestal veel objecten). In het werken met DSL's zijn de datacontracten een zeer belangrijk onderdeel omdat de meeste andere onderdelen in de architectuur er gebruik van maken. In de DTO's worden niet alleen de property's vastgelegd maar ook aan welke constraints deze moeten voldoen. De constraints kunnen weer gebruikt worden voor validatie van DTO. De constraints zijn gelimiteerd tot wat je in een XSD-schema kunt specificeren.

Businesslaag Zoals de figuur duidelijk maakt, bestaat de business-laag uit meerdere onderdelen. De business wordt altijd benaderd via een service interface. De service interface is een façade voor de service en zorgt voor authenticatie, logging en foutafhandeling. De service interface zelf bevat geen business logica en zal een verzoek dan ook altijd delegeren naar een business workflow of business proces. De service interface is stateless en daarmee zeer goed schaalbaar. De service interface zijn typisch ASMX-bestanden of classes met een service contract voor Windows Communication Foundation in .NET 3.0 en voldoen volledig aan de open standaarden voor webservices. Voor complexe, langdurige business-processen of processen waarbij externe services nodig zijn, wordt een business workflow gebruikt. De business workflow kan het beste worden ingevuld met de sequential workflows van Windows Workflow Foundation (WWF). Deze verzorgt zaken als het managen van langdurige transacties. Voor de aanroep van externe services in een business workflow zal bij voorkeur een service-agent gebruikt worden. Voor processen die direct afgehandeld kunnen worden wordt gebruik gemaakt van business processes. Deze zijn verantwoordelijk voor transactie management, mapping tussen DTO en business classes, en business class overstijgende logica. Business processen moeten ook kunnen participeren in een business workflow hetgeen betekent dat ze geschikt dienen te zijn voor gebruik binnen Windows Workflow Foundation. Het laatste onderdeel in de business laag zijn de business classes. Dit is een implementatie van het domeinmodel. Elke business class is verantwoordelijk voor relaties met andere business classes, validatie en business rules. Ophalen en persisteren van business classes gebeurt door een business proces dat daarbij gebruik maakt van de datalaag.

> De Microsoft referentie-architectuur laat over het algemeen nog veel ruimte over voor de detail-invulling

Datalaag De datalaag verzorgt het ophalen en persisteren van data. Voor alle business classes is er de mogelijkheid om deze op te halen en te persisteren. Wanneer data alleen getoond hoeven te worden en niet aangepast kunnen worden (bijvoorbeeld voor lijsten) is het ook mogelijk om DTO's rechtstreeks uit de datalaag te instantiëren. De datalaag is typisch een dunne laag die meestal gebruik maakt van OR-mapper library. Binnen SMART-Microsoft gebruiken we hier NHibernate voor. Indien data niet in een eigen database staat maar via een service uit een ander systeem komen, gaat dit via een data service-agent. Achter de datalaag wordt de

echte opslag van objecten in een traditionele relationele database gedaan.

Utility's Net als in de Microsoft referentie architectuur is er naast de lagen een verzameling van ondersteunende utility's die door alle lagen gebruikt wordt. Hier bevinden zich zaken als logging, configuratie en beveiliging. Deze utility's worden standaard ingevuld met de Enterprise Library van Microsoft. Verder is er per DSL een ondersteunend framework dat er maximaal op gericht is om her-generatie van code vanuit DSL-modellen mogelijk te maken. Deze frameworks zorgen ervoor dat de code die gegenereerd wordt op de juiste plaatsen aanpasbaar of uitbreidbaar is. Het is namelijk van cruciaal belang in een software factory dat de DSL-modellen altijd leidend blijven.

Ontwerp Bij het definiëren van de DSL's hebben we enkele uitgangspunten geformuleerd. Deze zijn leidend geweest bij de keuze voor de verschillende DSL's en de specifieke invulling van de DSL's zelf.

Modellen dienen eenvoudiger te zijn dan de equivalente code

Een model dient zich op een hoger abstractieniveau te bevinden dan de gegenereerde code. Dit betekent dat alleen concepten die in een model sneller en eenvoudiger te modelleren onderdeel van de DSL worden. Zaken die net zoveel werk kosten om te modelleren als te coderen, worden gewoon gecodeerd. Een direct gevolg hiervan is dat we niet per se volledige code-generatie ten doel hebben. De opzet van de DSL's en de bijbehorende code-generatie is, onder meer met behulp van partial classes in C#, zodanig dat handgeschreven code eenvoudig toe te voegen is aan de gegenereerde code.

DSL's moeten productief zijn voor ontwikkelaars met minder ervaring

Eén van de doelen van de SMART-Microsoft Software Factory is om ontwikkelaars met minder ervaring en minder diepgaande kennis van architectuur productief te maken. Dit bereiken we doordat de modellen technische details verbergen en derhalve eenvoudiger te ontwikkelen zijn dan de bijbehorende code. Modellen dienen derhalve relatief eenvoudig te blijven. Dit bereiken we door meerdere kleinere DSL's te definiëren. Per DSL is dan ook een beperkt aantal concepten beschikbaar voor de modelleur. Verder werken we binnen een DSL met meerdere kleine modellen. Dit maakt de modellen overzichtelijker en eenvoudiger te begrijpen.

Gegenereerde code dient leesbaar en onderhoudbaar te zijn

De relatie tussen de modellen en de gegenereerde code dient duidelijk te zijn voor de meer ervaren ontwikkelaars en architecten. Om deze reden hebben we ervoor gekozen om de DSL's te definiëren op basis van de architectuur, waarbij iedere DSL een expliciete relatie heeft met een of meer onderdelen uit de architectuur.

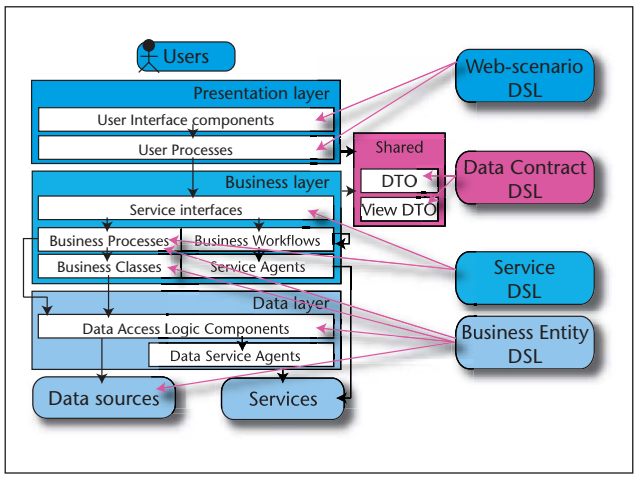
Hergeneratie moet altijd mogelijk zijn

Het genereren van code heeft alleen zin als het genereren op ieder moment in de tijd opnieuw gedaan kan worden. Hierbij moeten alle handmatige toevoegingen aan de code gegarandeerd behouden blijven. Deze doelstelling heeft directe gevolgen voor de structuur van de uit de DSL's gegenereerde code. We maken hierbij gebruik van een framework, en technieken als pattern, virtuele operaties of partial classes. Hoewel dit niet specifiek is voor de definitie van de DSL's is dit wel een voorwaarde om zinvolle DSL's te kunnen maken.

DSL's Voor de SMART-Microsoft Software Factory hebben we momenteel vier DSL's gedefinieerd. Iedere DSL kan worden afgebeeld op één of meer onderdelen uit de architectuur. Dit wordt getoond in figuur 2.

Webscenario DSL De webscenario DSL wordt gebruikt voor het modelleren van de presentatielaag. Er is specifiek gekozen om de webscenario DSL te maken voor webinterfaces. Als alternatief is bekeken of we een generieke DSL voor de gebruikerslaag zouden definiëren, die zowel geschikt zou zijn voor windows-interfaces als web-interfaces. Nadere analyse gaf duidelijk aan dat de structuur van de gebruikersinterface sterk beïnvloed wordt door de keuze tussen Web en Windows. Omdat het doel van de DSL is om code te genereren was de keuze voor een specifieke webscenario DSL de beste optie. Op deze wijze sluiten de concepten waarin gemodelleerd wordt zo goed mogelijk aan bij wat er daadwerkelijk door de gebruiker ervaren wordt en wat er gebouwd moet worden. De DSL is hierdoor gespecialiseerd voor web-interfaces.

Een kernconcept in deze DSL is de user action, het-



Figuur 2. DSL's en afbeelding naar de architectuur.

geen een combinatie is van het tonen van een webpagina en het uitvoeren van een actie door de gebruiker. Figuur 3 toont een voorbeeld van een webscenario-model. Hierin is Search Orders een user action van type List, welke een referentie bevat naar OrderDTO, een modelelement dat gedefinieerd wordt in een Data Contract model (zie figuur 4). De Edit order user action heeft als type Edit Action en refereert tevens naar een OrderDTO. Op basis hiervan worden complete ASP pagina's gegenereerd die geschikt zijn voor het werken met respectievelijk lijsten van objecten, of voor het bewerken van één enkel object. Voor de conditie Is Order open wordt een skeleton C# methode gegenereerd. De methode wordt binnen de gegenereerde code automatisch aangeroepen wanneer de gebruiker betreffende pad in het webscenario kiest. De ontwikkelaar werkt de conditie in C# verder uit. Het element Add Product is een verwijzing naar een ander webscenario, dat in een separaat model uitgewerkt is.

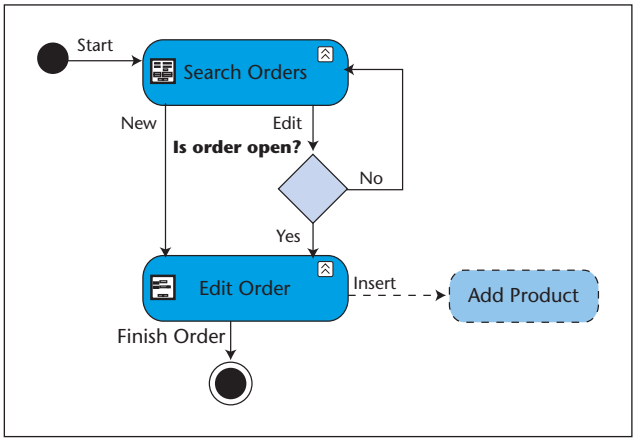
Naast de webscenario-DSL plannen we om een aparte Windows Presentation Foundation DSL te ontwikkelen, welke als alternatief voor de Web Scenario Designer kan dienen. Binnen dezelfde architectuur kan er dan gekozen worden voor verschillende invullingen van de presentatielaag.

Data Contract DSL Met de Data Contract DSL worden de Data Transfer Objecten (DTO) gemodelleerd. Deze DSL stelt ons in staat om alle data-objecten te definiëren die in de architectuur gebruikt worden voor de communicatie tussen de verschillende onderdelen in de architectuur.

In figuur 4 zijn verschillende types DTO-objecten gemodelleerd. OrderDto, ProductDto, CustomerDto en OrderLineDto zijn allen business DTO's, dat wil zeggen dat ze de data-representatie van een business object vormen. Een view DTO, zoals CustomerOrdersDto wordt gebruikt voor het modelleren van lijsten. De ProductDescriptionDTO is een filter DTO en representeert een beperkte kijk op de attributen van een business DTO. In het voorbeeld staat ook nog een composite DTO genaamd OrderOrderlineDTO. Deze DTO is een samenstelling van een OrderDTO en zijn bijgehorende OrderlineDTO's.

Service DSL SMART-Microsoft kent een service georiënteerde architectuur. We hebben derhalve een DSL nodig om services te modelleren. De Service DSL is gedefinieerd om alle benodigde service-interfaces te kunnen modelleren. De parameters van een service zijn data objecten en worden in de Data Contract DSL gemodelleerd. Vanuit de Service DSL wordt met behulp van referenties aangegeven welke Data Transfer Objecten als parameter gebruikt worden.

Bij de code-generatie worden, overeenkomend met



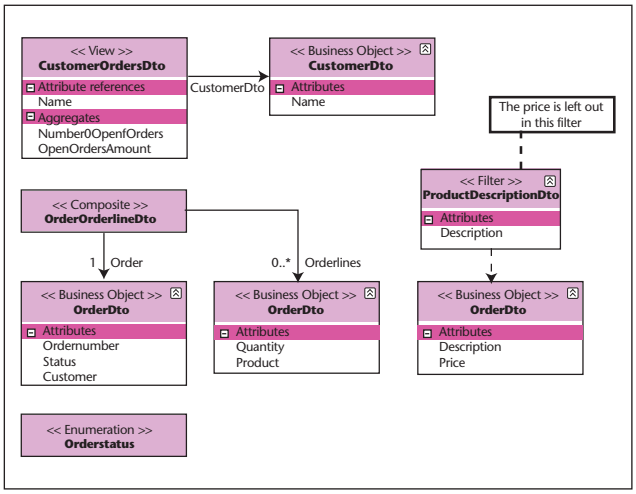
Figuur 3. Voorbeeld van Web Scenario Model.

de architectuur, zowel de service interfaces gegenereerd, alsook de skeletons voor business processen die betreffende services implementeren. Voor een aantal standaard CRUD (Create, Read Update, Delete) services wordt ook de implementatie van de service gegenereerd. De daadwerkelijke implementatie van de andere services is specifiek voor de betreffende business en niet

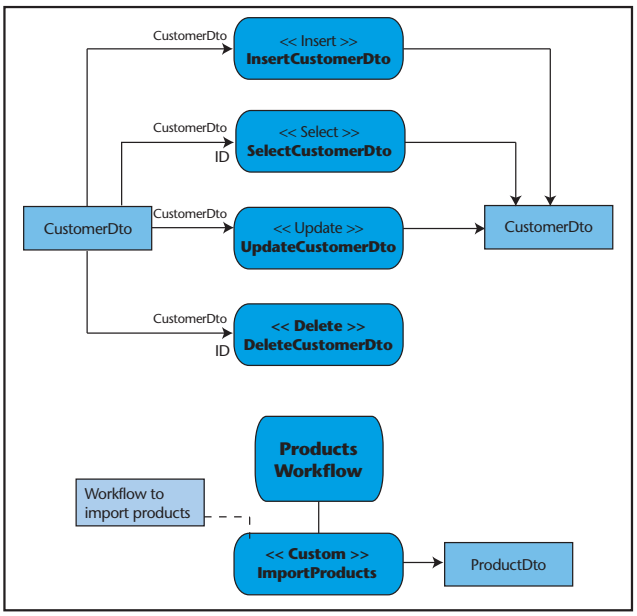
De transparantie zorgt ervoor dat de uit de modellen gegenereerde code te allen tijde toegankelijk blijft

gemodelleerd. Voor deze services wordt gebruik gemaakt van partial classes zodat de ontwikkelaar ze verder in C# kan schrijven.

In figuur 5 zijn de vier CRUD services gedefinieerd met de types <<Insert>>, <<Select>>, <<Update>>, <<Delete>>. De input en output parameters in de service DSL zijn referenties naar de in een Data Contract model gedefinieerde DTO's. Naast deze standaard services kunnen ook custom services, zoals ImportProducts



Figuur 4. Voorbeeld van Data Contract Model.

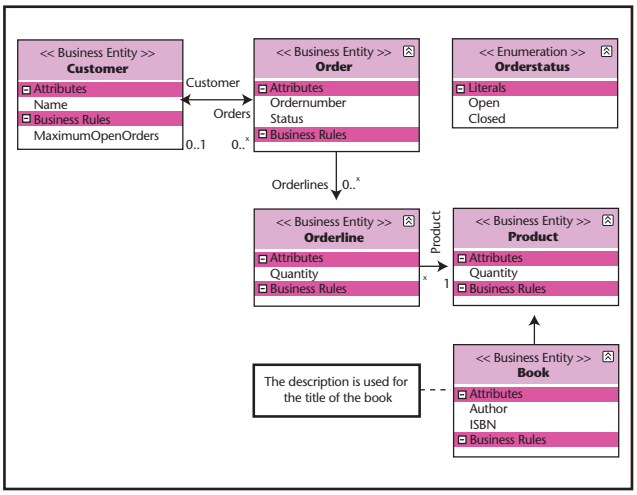


FIGUUR 5. Voorbeeld Service Model.

gedefinieerd worden. ImportProducts is in dit model bovendien enabled om in een workflow gebruikt te kunnen worden.

BUSINESS ENTITY DESIGNER De Business entity DSL stelt de ontwikkelaar in staat om business classes, inclusief hun attributen en onderlinge relaties, te definiëren. Vanuit de Business Entity DSL wordt code voor de Business Class laag gegenereerd, en tevens de volledige code voor de datalaag.

In figuur 6 zijn vijf business entity's gedefinieerd. Naast de attributen en relaties heeft Customer ook nog een business rule MaximumOpenOrders. Voor deze business rule wordt een skeleton methode gegenereerd, en het validatie framework zorgt ervoor dat deze methode op de juiste plaats en op het juiste moment wordt aangeroepen. De ontwikkelaar hoeft alleen nog maar de implementatie van deze methode te verzorgen.



FIGUUR 6. Voorbeeld Business Entity Model.

In de Business Entity DSL worden verder geen operaties of methodes op de business objecten gedefinieerd. De reden hiervan is eenvoudig. De enige code die we uit een methode definitie kunnen genereren is een één-op-één equivalente lagen methode in de C# code. Het schrijven van de methode in C# zelf is net zoveel werk. Aangezien we als uitgangspunt hebben dat er modelleren werk moet schelen is er vooralsnog weinig reden om dit in de Business entity DSL op te nemen. Methodes bij business objecten worden met behulp van partial classes in C# geschreven.

CONCLUSIE Omdat we meerdere DSL's definiëren voor verschillende onderdelen van de architectuur, betekent dit dat er bij het ontwikkelen van een applicatie meerdere modellen, behorende bij alle DSL's gemaakt worden. Tussen deze modellen bestaan vanzelfsprekend relaties, welke zich bevinden op het raakvlak van de architectuurlagen. Op deze wijze vertaalt de architectuur zich in verschillende DSL's en, omgekeerd, vertalen de verschillende DSL's zich naar de verschillende onderdelen in de lagen van de architectuur.

Deze transparantie zorgt ervoor dat de uit de modellen gegenereerde code te allen tijde toegankelijk blijft. Dit is van belang omdat niet alle onderdelen van een applicatie uit de modellen gegenereerd worden, een deel van de applicatie wordt rechtstreeks in C# geschreven. De plaatsen waar C# code toegevoegd wordt is expliciet gedefinieerd, zoals bijvoorbeeld de implementatie van de business rules of de service implementaties. De structuur van de gegenereerde code zorgt ervoor dat er te allen tijde opnieuw code gegenereerd kan worden uit de modellen, waarbij alle handmatige toevoegingen behouden blijven. De ontwikkelaar werkt derhalve continu in een combinatie van DSL's en C#.

Omdat de verschillende DSL's los van elkaar staan is het ook mogelijk om voor een specifiek project een deelverzameling van de DSL's te gebruiken. Zo kan bijvoorbeeld een applicatie welke geen webinterface kent alleen de webscenario DSL niet gebruiken, maar alle andere DSL's wel. In een ander geval kan een applicatie die aansluit op bestaande services alleen de Data Contract DSK en de Web Scenario DSL's gebruiken. Gebruik van de SMART-Microsoft Software Factory is derhalve geen alles-of-niets keuze. Dat komt de inzetbaarheid alleen maar ten goede.

Jos Warmer (jos.warmer@ordina.nl) is partner bij Ordina en Leendert Versluijs (leendert.versluijs@ordina.nl) is architect bij Ordina.