

Henri Ford in software- engineering

Op weg naar systematische softwarevariatie

Henri Ford wist het al: de grootste verbeteringen van het productieproces zijn te behalen door standaardisatie en herhaling. In hoeverre is dit inzicht ook van toepassing op de productie van software? Deze vraag is actueel in het software-engineeringvakgebied 'generatief programmeren', waarin bestudeerd wordt hoe een scala aan gespecialiseerde softwareproducten gegenereerd kan worden uit een als lopende band functionerende herbruikbare-componenteninfrastructuur.

Een voorbeeld van een generatieve oplossing is bij Fortis en ING. In samenwerking met het CWI en Cap Gemini is een gespecialiseerde taal (Risla) ontwikkeld om renteproducten te beschrijven. Uit deze beschrijvingen wordt onder meer Cobol-code gegenereerd voor de financiële administratie en systemen voor risicobeheer. De gegenereerde code maakt gebruik van een onderliggende componentenbiblio-

Domeinspecifieke talen worden ontwikkeld voor een specifiek gebied, bijvoorbeeld renteproducten, of telecommunicatie. Met een domeinspecifieke taal kan code worden gegenereerd voor verschillende softwarevarianten binnen een bepaald domein.

Arie van Deursen en Paul Klint

theek. Dankzij de toepassing van Risla is de *time-to-market* van nieuwe financiële producten teruggebracht van drie maanden tot drie weken. In het telecommunicatiedomein gebruikt Lucent Technologies de taal PRL5 om de integriteit van geconfigureerde telefooncentrales van het type 5ESS te verifiëren. Het hart van de 5ESS-software wordt gevormd door een relationele database met daarin informatie over hardware, softwareconfiguratie en klantspecifieke aanpassingen. De centrale werkt alleen als deze database voldoet aan een reeks van integriteitseisen. Deze eisen zijn in PRL5 declaratief te formuleren en worden gebruikt om diverse audits op de data uit te voeren en om transacties te controleren. Een derde voorbeeld is in gebruik bij de Navo. Hier wordt een taal

gebruikt om het formaat van boodschappen tussen tanks, vliegtuigen, en commandocentrales te beschrijven. Uit deze beschrijvingen wordt Ada-code gegenereerd om ontvangen en te versturen berichten te controleren op correctheid. Bovendien wordt SQL-code gegenereerd om databases te actualiseren, afhankelijk van de inhoud van de berichten. De bijbehorende taal is zo ontworpen dat deze aansluit op de werkwijze van de *message engineers* die verantwoordelijk zijn voor het ontwerpen van de berichtenstructuur. Deze lijst van voorbeelden kan moeiteloos worden uitgebreid met andere systemen. Bekijken we deze voorbeelden in meer detail dan valt op dat een gespecialiseerde taal daarin steeds een centrale rol speelt. Een dergelijke kleine, relatief eenvoudige taal wordt in de literatuur

Samenvatting

In het vakgebied 'generatief programmeren' wordt onderzocht hoe gespecialiseerde software-producten kunnen worden gegenereerd uit een infrastructuur van herbruikbare componenten. Herhaalbare acties worden gestandaardiseerd, waarmee besparingen kunnen worden behaald. Het doel is een productielijn voor software: een reeks op elkaar lijkende maar toch verschillende producten.

een domeinspecifieke taal (domain specific language: DSL) genoemd. Een DSL is wat notatie en concepten betreft toegespitst op een onderliggend applicatiedomein.

Een DSL heeft onder meer de volgende kenmerken.

- Een DSL bevat aanmerkelijk minder taalconstructies dan algemene programmeertalen zoals Cobol of Java.
- Een DSL staat dicht bij domeinexperts en ondersteunt vakspecifieke notatie en uitdrukkingwijzen.
- Een DSL is declaratief, en legt de nadruk op het 'wat' in plaats van het 'hoe'.
- De implementatie van domeinspecifieke talen is gebaseerd op codegeneratie, bijvoorbeeld naar C, Cobol, of Java. In veel gevallen zal de gegenereerde code aansluiten op een bijbehorende bibliotheek waarin domeinspecifieke concepten en procedures geïmplementeerd zijn.

Henri Fords principe van herhaling en standaardisatie wordt op deze manier toegepast op software-evolutie. Veel soorten wijzigingen hebben een systematisch karakter, zoals het toevoegen van een nieuw renteproduct, het introduceren van een nieuwe type bericht, of het verifiëren van de consistentie van een nieuwe configuratie van een telefooncentrale. In generatief programmeren wordt gepoogd deze herhaling te standaardiseren, door de introductie van een beschrijving op een hoger abstractieniveau, waaruit specifieke code gegenereerd kan worden.

De belangrijkste besparing ligt in de verminderde inspanning voor veelkomende wijzigingen, waardoor bijvoorbeeld sneller op de eisen van de markt ingespeeld kan worden. Het belangrijkste risico ligt in de investering voor de realisatie van de infrastructuur, die is gebaseerd op generatortechnologie. Het huidige onderzoek op het gebied van generatief programmeren (zie kader) probeert deze voor- en nadelen beter in kaart te brengen, en beoogt methoden en technieken te ontwikkelen waarmee de voordelen versterkt en de nadelen verholpen kunnen worden. In het bijzonder wordt gekeken naar een systematische methode om tot een succesvolle DSL te komen, en naar flexibele generatortechnologie waarmee generatieve oplossingen geïmplementeerd kunnen worden.

Methodologie

Essentieel voor het realiseren van een significante besparing is het herkennen van situaties waarin herhaling en standaardisatie in onderhoud een rol kunnen spelen. Hiertoe dienen de overeenkomsten en verschillen (de variabiliteit) tussen meerdere versies van dezelfde software vergeleken te worden. Merk op dat het hier niet alleen om evolutie van een enkel maatwerksysteem hoeft te gaan. Variabiliteit is juist ook van groot belang bij productsoftware, waar uit een product met klantspecifieke aanpassingen en configuratie meerdere gespecialiseerde systemen verkregen kunnen worden. In de voorbeelden ligt de variabili-

teit van de software in extra velden in de boodschappen tussen voertuigen, in extra afhankelijkheden die ontstaan door het toevoegen van functionaliteit in een telefooncentrale zoals bijvoorbeeld 'call waiting' en 'call forwarding', of in de invoering van een nieuw renteproduct, zoals een zeer specifiek 'schrikkeljaardeposito'. In elk domein moet een te ontwerpen DSL het mogelijk maken om al deze nieuwe eigenschappen uit te drukken. De variabiliteit mag echter niet onbeperkt zijn: de aard van de variabiliteit moet voorspelbaar zijn. Bovendien moet deze voorspelling enige tijd geldig blijven: de variabiliteit moet stabiel zijn. Zo is de aanname bij Risla dat er in de loop der tijd steeds meer renteproducten bij zullen komen, en dat die allemaal goed te beschrijven zijn, met behulp van zogenaamde 'cash flows'. Zowel in het onderzoek op het gebied van softwarehergebruik als op het gebied van telecommunicatie (in het bijzonder telefooncentrales) zijn methoden ontwikkeld om de variabiliteit in specifieke domeinen in kaart te brengen. Deze methoden vallen onder de noemer domeinengineering. Waar traditionele software-engineering zich richt op het bouwen van een individueel systeem, richt domeinengineering zich op het ontwikkelen van een familie van op elkaar lijkende systemen. Het doel is te komen tot een productielijn voor software: een versie van Henri Fords lopende band, waar door middel van eenvoudige configuratie een reeks op elkaar lijkende maar

toch verschillende softwareproducten gefabriceerd kunnen worden. Om de verschillen en overeenkomsten tussen de diverse systemen in de familie goed te onderscheiden wordt in de domeinengineering een zogenaamd *featuremodel* opgesteld, waarin de variabiliteit van het domein kan worden beschreven. Deze wordt uitgedrukt in termen van elementaire kenmerken (features), waarbij de nadruk wordt gelegd op de verschillen in kenmerken tussen de diverse systemen. Hierbij kan het ook gaan om gebruikersspecifieke varianten. Het geheel van kenmerken wordt uiteindelijk gecombineerd tot een grafisch kenmerkendiagram waarin verschillen en overeenkomsten in relatie tot elkaar worden weergegeven (zie kader). Is de variabiliteit eenmaal goed in kaart gebracht, dan kan een infrastructuur van herbruikbare componenten opgezet worden die configureerbaar is volgens de gevonden variabiliteit. Een domeinspecifieke taal is daarbij een hulpmiddel. Met een DSL kan een gewenste configuratie op hoog niveau beschreven worden. Uit deze beschrijving wordt code gegenereerd die elementen uit de domeinspecifieke bibliotheek aanroept.

Technologie

De technologische ondersteuning voor generatief programmeren is drieledig. Allereerst is het nodig een domeinspecifieke taal te ontwerpen. Hiervoor moeten we vorm

Generatief programmeren

Generatief programmeren richt zich op het ontwikkelen van families van softwaresystemen. Generatief programmeren bouwt voort op eerder onderzoek op het gebied van softwarehergebruik en programmageneratoren. Generatieve technieken spelen ook een belangrijke rol in architecturen voor softwareproductlijn.

Er worden diverse gespecialiseerde congressen en workshops op het gebied van generatief programmeren georganiseerd, waaronder de jaarlijkse conferentie over 'Generative and Component-Based Software-engineering' (GCSE) en 'Domain-Specific Languages for Software-engineering' (onderdeel van HICCS). Een belangrijke impuls voor dit onderzoeksgebied vormde het in 2000 verschenen boek van Czarnecki en Eisenecker.

In Nederland wordt op diverse universiteiten onderzoek verricht op gebieden gerelateerd aan dat van generatief programmeren. Dit artikel is geschreven op basis van ervaringen in het project 'Domain-Specific Languages' dat voor het Telematica Instituut is uitgevoerd. In dit project zijn de volgende resultaten bereikt (zie ook www.cwi.nl/projects/dsl).

1. De methodologie voor domain engineering is versterkt door hierin ook geautomatiseerde analyse van legacysystemen in op te nemen. Bovendien is de beschrijving van featuremodellen vereenvoudigd met de 'Feature Description Language' FDL.
2. De basistechnologie voor het ontwerpen van DSLs en voor het genereren van tools is verder ontwikkeld. Deze is beschikbaar als de 'ASF+SDF Meta-Environment'.
3. Een aantal case study's heeft de bruikbaarheid van de DSL-benadering aangetoond.

(syntax) en eventuele randvoorwaarden vastleggen. In praktische zin betekent dit het opstellen van een grammatica en het formeel specificeren van de randvoorwaarden. Een belangrijk uitgangspunt bij dit ontwerp zijn vaak al aanwezige bibliotheken die de essentiële begrippen van het domein al implementeren.

Ten tweede is het nodig om programma's in de domeinspecifieke taal af te beelden op het onderliggende hard- en softwareplatform en te integreren met de aanwezige bibliotheken. Dit komt neer op het bouwen van een codegenerator van de domeinspecifieke taal naar bijvoorbeeld Cobol of Java. De complexiteit van de generator hangt af van de mate waarin het nodig is gegenereerde code geautomatiseerd te 'verweven' met handgeschreven code. Dit kan bijvoorbeeld helpen om code voor moeilijk te modulariseren aspecten (zoals foutafhandeling of tracing) te integreren met aanwezige code.

Ten slotte is het nodig om de domeinspecifieke taal en alle daarin geschreven programma's te ondersteunen tijdens hun verdere evolutie. Als er bijvoorbeeld wijzigingen in de taal nodig zijn, dan impliceert dit wijziging van de codegenerator en eventuele migratie van alle aanwezige programma's.

Perspectief

Generatief programmeren en domeinspecifieke talen maken het mogelijk om een reeks van systeemvarianten eenvoudig af te leiden uit een herbruikbare-componenteninfrastructuur. Generatief programmeren en domeinspecifieke talen vormen een actief onderzoeksgebied, waarin thema's zoals *aspect oriented programming*, codegeneratoren, en software-productlijnen een belangrijke rol spelen. Deze thema's worden niet alleen aan universiteiten, maar ook in diverse industriële laboratoria onderzocht, bijvoorbeeld bij Philips en Nokia.

Domeinspecifieke talen en generatief programmeren vormen tevens een interessant perspectief op de tegenwoordige ontwikkelingen rondom XML. Veel DTD's (*document type definitions*) kunnen beschouwd worden als domeinspecifieke talen. Zowel bij het ontwerp van zo'n DTD, als bij de implementatie van tools om DTD-specifieke XML-documenten te manipuleren, kan vruchtbaar gebruik gemaakt worden van de methoden en technieken zoals generatief programmeren en domeinspecifieke talen.

Een ruwe indicatie voor het invoeren van een DSL is de 'regel van 3×3 ': als je drie maal een applicatie in een bepaald domein hebt gebouwd bezit je voldoende kennis om een domeinmodel te kunnen maken. De werkelijke uitwerking van dit model is vervolgens pas rendabel als je nog eens drie varianten in hetzelfde domein zult gaan ontwik-

kelen. Het domein moet daarbij voldoende stabiel zijn om de variabiliteit in kaart te kunnen brengen. Het gebruik van tools bij ontwerp, implementatie en evolutie is essentieel om de kosten van de te implementeren infrastructuur beperkt te houden.

De zo verkregen herbruikbare-componenteninfrastructuur is een belangrijke stap op weg naar het toepassen van Henri Fords inzichten: het door herhaling en stan-

daardisatie produceren van softwareproducten binnen hetzelfde toepassingsdomein.

Reviewer Jacques Vandenbulcke

Arie van Deursen

is projectleider bij het CWI en universitair hoofddocent Software-engineering bij de TuD.
E-mail: Arie.van.Deursen@cw.nl.

Paul Klint

is hoofd Software-engineering van het CWI en hoogleraar Informatica bij de UvA.
E-mail: Paul.Klint@cw.nl.

Literatuur

Czarnecki, K. & U.W. Eisenecker (2000). *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley.

Deursen, A. van, P. Klint, & J. Visser (2000). Domain-Specific Languages: An Annotated Bibliography. *ACM SIGPLAN Notices* 35(6):26-36, June 2000.

Links

www.program-transformation.org, waar onder meer de 'generative programming wiki' te vinden is met allerlei informatie op het gebied van generative programming

www.sei.cmu.edu/programs/pls/, de softwareproductlijnsite van het Software-engineering Institute (SEI).