Transparantie in variabiliteit

Heldere keuzes in een eenduidige configuratieomgeving

Elk eerbiedwaardig softwaresysteem biedt tegenwoordig de mogelijkheid om gedurende de levensduur de eigenschappen ervan te wijzigen. Om deze variatiepunten te modelleren wordt in het Jaquard-project *TraCE* gewerkt aan transparante configuratieomgevingen.

Eelco Dolstra, Gert Florijn, Merijn de Jonge en Eelco Visser

In plaats van het ontwikkelen en uitleveren van één enkele versie van een product (*one-of-a-kind system*) is het tegenwoordig gebruikelijk een familie van systemen te ontwikkelen. In navolging van andere industrieën wordt dit een productlijn genoemd. De producten in een productlijn verschillen onderling in functionaliteit en technische voorzieningen. Kantoorapplicaties zoals tekstverwerkers worden vaak in verschillende varianten aangeboden, bijvoorbeeld in een standaard, professionele of ondernemingsversie. Deze varianten verschillen in prijs en in functionaliteit: hoe duurder, hoe meer functionaliteit. De mogelijkheid om de voor de gebruiker zichtbare eigenschappen (*features*) van een product te kiezen op enig moment gedurende de levensloop van een softwaresysteem wordt aangeduid met 'variabiliteit'. Andere typerende voorbeelden van variabiliteit zijn de noodzaak om software op verschillende platformen te kunnen draaien en om in verschillende natuurlijke talen met gebruikers te communiceren. De Linux-*kernel* – de basis van verschillende besturingssystemen zoals Red Hat Linux en SuSE Linux – geeft een goed voorbeeld van variabiliteit in moderne softwaresystemen.

Een belangrijke reden voor het ondersteunen van productvariaties is om hergebruik van systeemonderdelen te bevorderen. Als voor elke variant een afzonderlijk systeem zou moeten worden ontwikkeld, zou de totale ontwikkelinspanning sterk toenemen en zou de *time-to-market* negatief beïnvloed worden. Daarnaast zouden de onderhoudskosten en -inspanningen sterk toenemen omdat onderhoud aan elk van de systemen herhaaldelijk moet worden uitgevoerd wanneer zij overlappende functionaliteit bevatten.

Variatiepunten

Een productlijn bestaat uit aspecten die gemeenschappelijk zijn voor alle producten (*commonality's*) en aspecten die kunnen variëren. Bij elk variabel aspect is een keuze mogelijk tussen een aantal varianten. Zo'n aspect heet een variatiepunt. Recente versies van de Linux-kernel hebben meer dan 1500 expliciete variatiepunten. Voorbeelden hiervan zijn ondersteuning voor een bepaald doelplatform (bijvoorbeeld Intel x86), een single- of multiprocessorsysteem of netwerkprotocollen en filesystemen. Ook kan een variatiepunt betrekking hebben op bepaalde soorten harde schijven of netwerkkaarten. Zulke variatiepunten hebben weer allerlei parameters die het gedrag verder afstemmen, zoals het maximale aantal schijven dat wordt ondersteund door de scsi-diskdriver. De kernelbroncodedistributie vormt dus een productlijn met een zeer groot aantal mogelijke producten. Als we voor

Samenvatting

Steeds vaker zijn producteigenschappen van softwaresystemen op verschillende momenten te selecteren: tijdens het bouwen, het installeren, of de executie van het systeem. In het Jaquard-project TraCE werken de auteurs aan de ontwikkeling van transparante configuratieomgevingen. Hierin kan variabiliteit in een systeem worden gemodelleerd onafhankelijk van de implementatietechniek.

het gemak aannemen dat alle variatiepunten twee mogelijkheden bevatten, komen we op meer dan 10^{480} varianten. Aangezien Linux een open-sourcesysteem is, kunnen gebruikers het systeem daadwerkelijk binnen deze variabiliteitsruimte aanpassen aan de eigen wensen.

Featuremodellen

Een gebruikelijke aanpak in de ontwikkeling van productlijnen is het identificeren van features en hun onderlinge relaties in een featuremodel. Een featuremodel specificeert de variabiliteit van een systeem op conceptueel niveau. Het vormt de basis voor het ontwerp en de implementatie van gemeenschappelijke en productspecifieke onderdelen. Met het model kan men bij het ontwerp en de implementatie van het systeem de juiste abstracties inbouwen om met toekomstige varianten om te gaan. Echter, omdat featuremodellen geen automatisch onderdeel zijn van ontwikkelomgevingen, moet een ontwikkelaar de relatie tussen variabiliteit op het conceptuele niveau en de concrete implementatie ervan handmatig bijhouden. Deze relatie is vaak niet expliciet aanwezig en neigt te verdwijnen naarmate meer onderhoud gepleegd wordt.
Het is daarom goed om featuremodellen direct in de implementatie in te zetten. De Linux-kernel is hiervan een goed voorbeeld. Het heeft een formeel featuremodel dat de features en de beperkingen ertussen codeert. Een goed voorbeeld is de volgende afhankelijkheid: 'ondersteuning voor scsi-tapedrives is alleen beschikbaar als ondersteuning voor scsi-apparatuur aanstaat'. Een voordeel van zo'n model is dat het kan worden gebruikt om het configuratieproces te ondersteunen. De kernel beschikt over een grafische tool waarin features interactief geselecteerd kunnen worden, waarbij de consistentie van het geheel automatisch wordt gecontroleerd. Uit deze featureselectie kan vervolgens het gewenste product van de productfamilie geheel automatisch worden gegenereerd.

Tijdlijn

Een featuremodel op zich is niet genoeg om een implementatie te verkrijgen, omdat de tijdsaspecten van de variatiepunten een belangrijke rol spelen bij het ontwerp van een systeem. Om variabiliteit te implementeren is het nodig om voor elk variatiepunt te beslissen hoe en wanneer er een concrete variant kan worden geselecteerd. Dit selecteren heet binding van variatiepunten. Voorbeelden van de momenten waarop gebonden wordt zijn distributietijd, bouwtijd, installatietijd, opstarttijd en executietijd. De mogelijke belanghebbenden (*stakeholders*) die dit soort keuzes maken (ontwerpers, ontwikkelaars, productmanagers, eindgebruikers) hebben invloed op de uiteindelijke verzameling keuzemomenten van een softwaresysteem.
De eenvoudigste beslissing is om alle variatiepunten op hetzelfde moment en via hetzelfde mechanisme te implementeren. De Linux-kernel was oorspronkelijk geïmplementeerd als een traditionele monolithische kernel. Dit wil zeggen dat alle functionaliteit zoals drivers voor randapparatuur statisch in de kernel zit. Conditionele *defines* en manipulatie van *makefiles* worden gebruikt om features selectief tijdens bouwtijd in of uit te schakelen. Vrijwel alle variatiepunten worden dus tijdens het bouwen gebonden.
Het nadeel van deze aanpak is dat het een groot aantal variatiepunten na de bouwtijd verdwenen zijn. Een gevolg is dat bijvoorbeeld het installeren van een nieuwe driver een hercompilatie van de kernel vereist. Daarom werd de kernel later voorzien van een modulesysteem. Een verzameling broncodebestanden die samen een module vormen, kunnen tot een enkel objectbestand worden gecompileerd en ofwel statisch worden meegelinkt met de kernel, of dynamisch in een draaiend systeem worden ingeladen. Modules kunnen verwijzen naar andere modules; bij het inladen van een module zorgt een tool ervoor dat alle modules waarnaar verwezen wordt automatisch in de juiste volgorde worden meegeladen.

Het cruciale punt is dat modules zowel tijdens bouwtijd als gedurende de uitvoering kunnen worden 'ingeschakeld'. Het verschijnsel dat een systeem meer momenten heeft waarop variatiepunten gebonden kunnen worden heet *tijdlijnvariabiliteit*. Dit is een extra dimensie van variabiliteit die meestal wordt genegeerd. Tijdlijnaspecten zoals keuzemomenten en keuzemakers worden dan ook meestal niet gemodelleerd. Een gevolg hiervan is dat variabiliteit niet op de tijdlijn staat aangegeven en dat variabiliteit vaak ad hoc ontworpen is. Sommige keuzes kunnen tijdens de compilatie gemaakt worden, andere tijdens de installatie en weer andere tijdens het draaien. Daarnaast staat het keuzemoment van een variatiepunt meestal vast en kan deze niet meer gewijzigd worden. Een installatietijdvariatiepunt kan dan niet op compilatietijd gebonden worden.

Implementatie

Tijdlijnvariabiliteit is lastig te implementeren. Dit is vrijwel onvermijdelijk omdat programmeertalen variabiliteit niet direct ondersteunen. Om een variatiepunt te kunnen laten binden ofwel op tijdstip X ofwel op tijdstip Y is relatief eenvoudig, maar om het zowel op X als op Y te kunnen binden vereist meestal enige trucs. Neem bij wijze van eenvoudig voorbeeld een tweeledig variatiepunt dat op *executietijd* gebonden wordt. Dit kan in de programmeertaal C triviaal worden geïmplementeerd zoals getoond in fragment 1.

```
if (feature) f()
else g();
```

Figuur 1. C-code tijdens executietijd gebonden tweeledig variatiepunt

Het is ook niet moeilijk dit variatiepunt tijdens de bouwtijd te laten binden.

```
#if FEATURE
  f()
#else
  g()
#endif
```

Figuur 2 C-code tijdens bouwtijd gebonden tweeledig variatiepunt

Echter, als we deze feature zowel tijdens bouwtijd als executietijd willen kunnen laten binden, wordt de zaak ingewikkelder. Zie hiervoor fragment 3.

```
#if BOUND_AT_BUILD_TIME
#if FEATURE f()
  #else g()
#endif
#else
  if (feature) f()
  else g()
#endif
```

Figuur 3 C-code zowel tijdens bouwtijd als executietijd te binden binair variatiepunt

Dit is niet erg aantrekkelijk omdat we nu met twee verschillende implementatiemechanismen te maken hebben. Het wijzigen van tijdsaspecten is dus veel werk. Dit wordt voor een belangrijk deel veroorzaakt doordat de implementatie van een variatiepunt verspreid is over veel onderdelen van een softwaresysteem zoals bijvoorbeeld broncode- en *build*files. Het toestaan van bijvoorbeeld het vooraf binden van installatietijdvariatiepunten kan veel aanpassingen vergen aan verschillende onderdelen van een softwaresysteem. Een andere eigenschap van variabiliteitsmechanismen is dat de consequenties van een specifieke keus voor een variatiemoment voor bijvoorbeeld performance, resourcegebruik of onderhoudbaarheid niet helder zijn. Hierdoor kunnen eventuele technieken om software op specifieke bindingstijden te optimaliseren niet ten volle benut worden.

Configuratie

Een ander probleem is dat variatiepunten via een of ander configuratiemechanisme gebonden moeten worden. Het probleem met systemen die configuratie op verschillende momenten op de tijdlijn toestaan is dat de configuratie-interface op elk conceptueel moment verschillend is. Zo kunnen in het geval van de Linux-kernel modules tijdens bouwtijd worden toegevoegd via een tool dat de gebruiker een grafisch configuratie-interface biedt om variatiepunten te binden. Tijdens de executie van het systeem is de configuratie-

interface echter aanzienlijk primitiever. Het toevoegen van een module geschiedt dan via commando's zoals *modprobe* die simpelweg de naam van een module als argument nemen. Er wordt geen featuremodel aan de gebruiker gepresenteerd.

Een softwaresysteem dat variatiemogelijkheden heeft, biedt een configuratie-interface waarmee keuzes mogelijk zijn. Deze interface correspondeert idealiter met de variabiliteit op het conceptuele niveau, dat wil zeggen met het featuremodel. Helaas wordt de interface doorgaans bepaald door het onderliggende variabiliteitsmechanisme. Zo wordt de abstracte blik op variabiliteit vertroebeld door implementatiemechanismen op laag niveau en bepaalt de techniek de te gebruiken configuratie-interface. Omdat variabiliteitskeuzen door verschillende mechanismen gerealiseerd worden, die elk sterk afhankelijk zijn van het moment op de tijdslijn, is er geen sprake van consistente configuratie-interfaces. Hierdoor wordt variabiliteit dus niet transparant maar afhankelijk van het variatiemoment.

TraCE

Om de hierboven beschreven problemen op te lossen, heeft het Jacquard-project *TraCE* (*Transparent Configuration Environments*) als doel een studie te verrichten naar de modellering en realisatie van variabiliteit in moderne softwaresystemen. De nadruk ligt daarbij op een meer algemene en generieke behandeling van tijdlijnaspecten van variabiliteit. In het project wordt een raamwerk ontwikkeld voor transparante configuratieomgevingen met een uniforme interface voor allerlei onderliggende configuratiemechanismen. Dit overbrugt de afstand tussen variabiliteit op het concept- en het implementatieniveau.
Een transparante configuratieomgeving is een tool voor het configureren van een softwaresysteem. Centraal in de TraCE-architectuur staat een taal waarmee variabiliteit in systemen formeel uitgedrukt kan worden, onafhankelijk van de concrete implementatiemechanismen. De tijdlijneigenschappen van de variatiepunten worden

»Variabele eigenschappen worden meestal ad hoc ontworpen«

uitgedrukt door overgangen die corresponderen met het maken van configuratiekeuzes. De generieke configuratietool TraCE vertaalt deze overgangen op het modelniveau naar concrete acties op het implementatieniveau. TraCE houdt de toestand van een softwaresysteem bij die de configuratietoestand van het systeem op elk tijdsmoment weergeeft. TraCE voert wijzigingen aan de configuratie vervolgens automatisch door in het systeem.

Literatuur

Bosch, J. (2000). *Design and Use of Software Architectures: Adopting and Evolving a Product-Line Approach*. Addison-Wesley.

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

Dolstra, E., G. Florijn, M. de Jonge & E. Visser (2003). *Capturing Timeline Variability with Transparent Configuration Environments. In International Workshop on Software Variability Management*. Portland, Oregon, USA, May 2003.

Raymond, E.S. (2001). The CML2 language: Python implementation of a constraint-based interactive configurator. In *9th International Python Conference, March 2001*.

Link

www.cs.uu.nl/group/ST/TraCE

Eelco Dolstra
is assistent in opleiding aan de Universiteit Utrecht.
E-mail: eelco@cs.uu.nl.

Gert Florijn
is managing consultant bij Cibit-Serc ICT adviseurs.
E-mail: florijn@serc.nl.

Merijn de Jonge
is postdoctoraal onderzoeker aan de Universiteit Utrecht.
E-mail: mdejonge@cs.uu.nl.

Eelco Visser
is docent-onderzoeker aan deUniversiteit Utrecht.
E-mail: visser@cs.uu.nl.