Intelligente levering van productsoftware

Kennisrepresentatie en webgebaseerde distributie

Het Jacquard-project *Deliver* is 1 juli 2003 van start gegaan en richt zich op de ontwikkeling van een intelligente softwarekennisbank om het configureren en distribueren van productsoftware te verbeteren.

Gerco Ballintijn, Sjaak Brinkkemper, Remy Jansen, Paul Klint en Tijs van der Storm

Voor leveranciers van softwareproducten wordt het steeds moeilijker om de softwareconfiguraties bij hun klanten te beheren en te controleren. De software draait op uiteenlopende hardware en software platforms en is vaak voor een specifieke klantinstallatie geparametriseerd of geoptimaliseerd. Op dit moment worden deze configuraties als gedetailleerde lijsten van software componenten semi-automatisch bijgehouden. Dit is arbeidsintensief en foutgevoelig. Om het beheren en actualiseren van softwareconfiguraties te vereenvoudigen stellen wij voor om een intelligente softwarekennisbank in te voeren die alle feiten over alle software componenten bevat samen met relevante attributen, onderlinge relaties en beperkingen (*constraints*). Op deze manier kunnen correcte softwareconfiguraties automatisch berekend worden gegeven een klein aantal sleutelparameters. Het wordt ook mogelijk om *wat-als*-vragen te stellen over potentiële wijzigingen van configuraties bij klanten.

Het beheren van de softwareconfiguraties is echter maar een deel van het probleem. Nieuwe of gewijzigde configuraties moeten ook nog bij de klant geïnstalleerd worden. Om dit te bereiken, is het nodig om het verschil te berekenen tussen een bestaande configuratie en een gewenste configuratie. De gevonden configuratieverschillen moeten vervolgens via speciale protocollen bij de klant afgeleverd worden. Vandaar de naam van het project: Deliver.

Het wetenschappelijke doel van dit project is om methoden en technieken te ontwikkelen voor het volledig automatisch controleren en via het web distribueren, integreren en opwaarderen van productsoftware.

Economische achtergrond

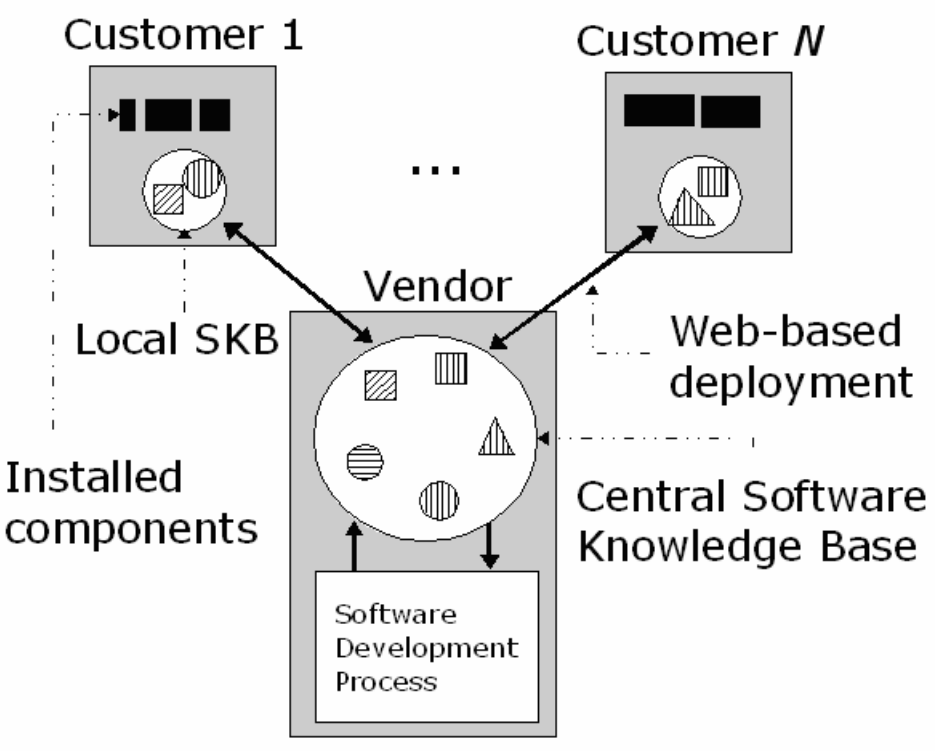
Het economische doel van dit project is om de productie van productsoftware in Nederland te versterken. Volgens studies van de Oeso vertegenwoordigt productsoftware wereldwijd een aanzienlijke economische waarde (Oeso, 2001). In 1999 werd de totale markt voor productsoftware geschat op 155 miljard dollar, met een jaarlijkse groei van 11%. In datzelfde jaar werd in Nederland voor 2,1 miljard euro aan productsoftware ingevoerd, terwijl de export 0,6 miljard euro bedroeg. Dit tekort op de softwarehandelsbalans is kenmerkend voor heel

Europa, en is een belangrijke indicator dat de Nederlandse industrie op het gebied van productsoftware versterkt moet worden.
Productsoftware kan onderverdeeld worden in een aantal categorieën. De belangrijkste zijn consumententoepassingen, softwareontwikkelgereedschappen, systeemsoftware en bedrijfstoepassingen. We gaan hier alleen op consumenten- en bedrijfstoepassingen in.
Consumentenproducten en thuistoepassingen staan ook wel bekend onder de naam 'shrink-wrap software'. MS Office is een typisch voorbeeld: een volledig standaardproduct dat alleen aangepast wordt voor nationale markten (taal, indeling toetsenbord). De klant kan deze producten via het Web aanschaffen en opwaarderen maar dit heeft verder geen invloed op de functionaliteit per pakket. Een representatief voorbeeld van bedrijfstoepassingen zijn systemen voor bedrijfsplanning (*enterprise resource planning*: erp) zoals bijvoorbeeld geleverd worden door Baan, Exact, Oracle, Sap en Peoplesoft. Elke leverancier heeft een verzameling standaardcomponenten die aangepast en gecombineerd worden om te voldoen aan de wensen van een specifieke klant. In dit geval is levering en opwaardering via het web ingewikkelder omdat iedere klant een unieke combinatie gebruikt van standaardcomponenten, aangepaste standaardcomponenten en speciaal voor de klant ontwikkelde componenten.

Techniek

Het is voor productsoftware in het algemeen en voor bedrijfstoepassingen in het bijzonder essentieel om voor elke klant nauwkeurig bij te houden welke versies van componenten in gebruik zijn. Om deze verschillende versies van componenten ook feitelijk te kunnen leveren, vereist op zijn beurt een nauwkeurige administratie van alle softwareartefacten die nodig zijn om deze componenten te bouwen, te distribueren en te installeren. Onder softwareartefacten verstaan we broncode, *include files*, scripts voor bouwen en configureren, revisiehistories, foutendatabases, applicatiebibliotheken, tests, testhistories, documentatie, helpbestanden en data (iconen, geluids- en videofragmenten). Het bijhouden van al deze artefacten is een groot probleem omdat de geïnstalleerde versie en de 'huidige' versie van elke component waarschijnlijk verschilt en vaak andere beperkingen oplegt aan de beschikbaarheid, stabiliteit en prestaties van andere componenten. Als we ook nog het aantal configuratieparameters op verschillende platforms in beschouwing nemen (hardware, besturingssysteem, databasesysteem, gebruikersinterface, middleware) dan wordt het al snel duidelijk dat het aantal mogelijke configuraties exponentieel toeneemt en dat het ondoenlijk is om alle configuraties ook daadwerkelijk te genereren. Het aantal aspecten dat per component een rol speelt is ook indrukwekkend.

- Welke artefacten zijn nodig voor deze component.
- Kan de component gecompileerd, getest en geïntegreerd worden.
- Voor welke software-/hardwareplatforms is de component beschikbaar.
- Van welke (versies van) andere componenten is de component afhankelijk.
- Welke beperkingen bestaan er op gebruik of integratie van de component.
- Welke (versies van) artefacten die nodig zijn voor deze component, zijn al geïnstalleerd bij de klant.



Figuur 1: Globale architectuur

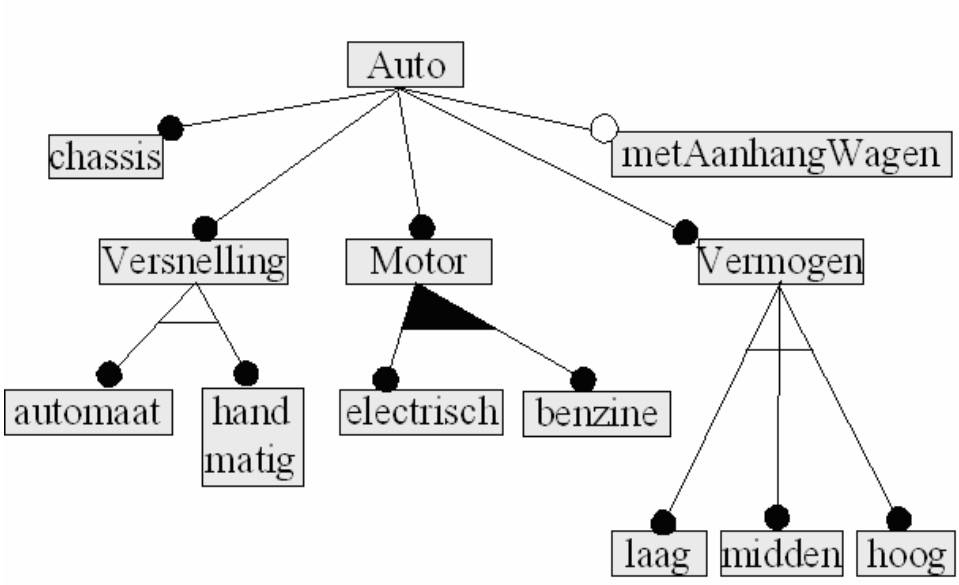
Het is gemakkelijk in te zien dat al deze factoren leiden tot een combinatorische explosie van mogelijke configuraties en dat een handmatige of semi-automatische administratie daarvan veel werk kost en foutgevoelig is. Hiermee krijgen we onvoldoende inzicht en is kwaliteitscontrole niet goed mogelijk.

Onderzoekslijnen

Om hier verandering in te brengen is het nodig om het proces van bouwen, distribueren en integreren inzichtelijker en beheersbaar te maken. Dit kan bereikt worden met een intelligente softwarekennisbank en een webgebaseerd distributieproces (zie figuur 1).

Een intelligente softwarekennisbank bevat uitputtende informatie over alle softwareartefacten en hun onderlinge verbanden en beperkingen. Bovendien bevat de kennisbank regels over het softwareproces: een component is bijvoorbeeld alleen maar beschikbaar als deze met succes getest is. Op deze manier, kan een gedetailleerde configuratie van de gebruiker berekend worden op basis van een klein aantal parameters. Dit is een volledig automatische methode die een gegarandeerde kwaliteit levert. De kennisbank is bovendien in staat om wat-als-vragen te stellen van de vorm ‘wat gebeurt er als we voor klant K component X opwaarderen van versie 6 naar versie 7 en de nieuwe component Y toevoegen?’ Op deze manier worden planning en kwaliteitscontrole mogelijk.

Een webgebaseerd distributieproces zorgt voor distributie, opwaardering en vervanging van softwarecomponenten. Met de informatie die de softwarekennisbank genereert kan de klantconfiguratie op afstand beheerd en aangepast worden.



Figuur 2: Featurediagram voor Auto

Aanpak

Om de hierboven geschetste situatie te bereiken is het nodig om kennis over software te representeren en om protocollen voor webgebaseerde distributie van software mogelijk te maken. Om de kennis over softwareartefacten vast te leggen valt als eerste te denken aan een relationele database om de onderlinge verbanden tussen artefacten in op te slaan. Sql kan dan gebruikt worden om analyses en wat-als-vragen te formuleren.

Om de variabiliteit van software te representeren is het echter nodig om keuzemogelijkheden vast te leggen en de beperkingen die aan die keuzes opgelegd moeten worden: bepaalde componenten sluiten elkaar uit of vergen juist de aanwezigheid van andere componenten in specifieke versies. Verder vraagt het analyseren van de verbanden tussen artefacten om indirecte (transitieve) relaties: de indirecte aanroeprelatie tussen componenten is bijvoorbeeld nodig om alle afhankelijkheden van een component te bepalen.

Omdat standaard relationele databases bovenstaande functionaliteit onvoldoende ondersteunen kiezen we een ander, abstracter, uitgangspunt voor de softwarekennisbank: featurediagrammen. Featurediagrammen zijn ontstaan in de telecomindustrie om greep te krijgen op de grote hoeveelheden features in telefooncentrales (Kang e.a., 1990). Meer recent worden ze ook toegepast om de features van software te modelleren (Czarnecky & Eisenecker, 2000; Van Deursen & Klint, 2002).

```
Car: all ( carBody, Transmission, Engine, HorsePower, PullsTrailer? )
Transmission: one-of ( automatic, manual )
Engine: more-of ( electric, gasoline )
HorsePower: one-of ( lowPower, mediumPower, highPower )
```

Figuur 3: FDL-versie van het featurediagram voor Auto

Figuur 2 geeft een featurediagram voor een eenvoudige auto en is ontleend aan Czarnecky & Eisenecker. Dit diagram legt vast dat een auto bestaat uit ‘chassis’, ‘Versnelling’, ‘Motor’ en ‘Vermogen’. Deze vier features zijn altijd nodig; dit wordt aangegeven door het opgevulde cirkeltje aan de bovenkant van elk feature. Het laatste feature van de auto is ‘metAanhangWagen’. Dit feature is optioneel en dit wordt aangegeven door het open cirkeltje. ‘chassis’ en ‘metAanhangWagen’ zijn atomaire features die niet verder opgedeeld kunnen worden in andere features. Features die gedefinieerd worden in termen van andere features noemen we ‘samengestelde features’.[1]

De versnelling kan ‘automatisch’ of ‘handmatig’ zijn. De open driehoek tussen ‘Versnelling’ en zijn subfeatures geeft aan dat dit een exclusieve keuze is (‘one-of’): hij is ‘automatisch’ of ‘handmatig’ maar niet allebei tegelijk.

De motor kan ‘elektrisch’ zijn of op ‘benzine’ lopen. De opgevulde driehoek geeft aan dat het een niet-exclusieve keuze is (‘more-of’): of elektrisch, of benzine, of beiden zijn mogelijk.

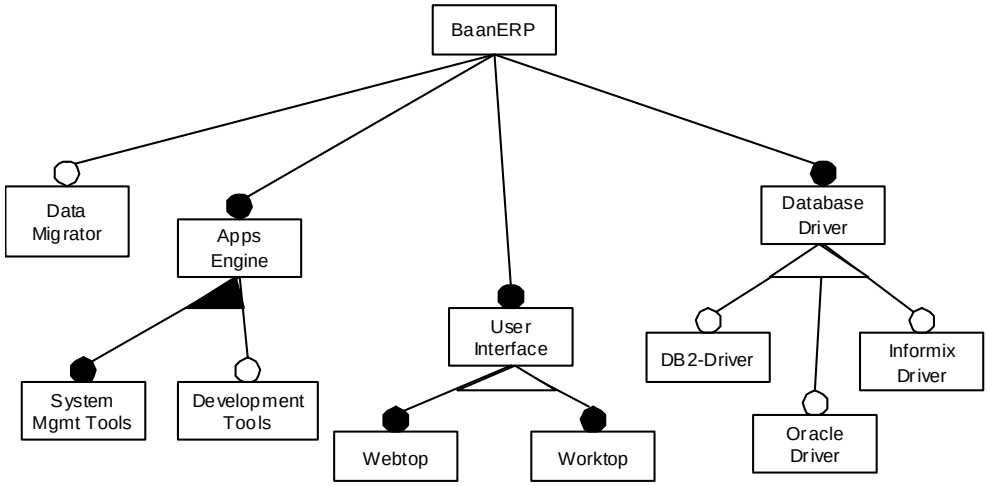
Het vermogen kan ‘laag’, ‘midden’ of ‘hoog’ zijn. De open driehoek geeft aan dat het een exclusieve keuze betreft.

Een instantie van een featurediagram bestaat uit een feitelijke keuze voor de atomaire features die voldoen aan de eisen die het diagram oplegt. Een instantie komt overeen met een productconfiguratie van een systeemfamilie. Analyse van dit voorbeeld leert dat het aantal mogelijke instanties in dit geval 36 is.

Het beschrijven van complexe producten met deze grafische notatie wordt al snel onoverzichtelijk. Daarom hebben wij in Van Deursen & Klint (1998) een equivalente tekstuele versie voorgesteld met de naam *Feature Description Language* (FDL). Een FDL-versie van ‘Auto’ is te zien in figuur 3.

Een belangrijk onderdeel van FDL zijn de beperkingen die aan een selectie van features opgelegd kunnen worden. Een kenmerkende beperking in het autovoorbeeld zou kunnen zijn ‘metAanhangWagen vereist hoogVermogen’. Dit heeft als effect dat alleen de ‘hoogVermogen’-variant geselecteerd kan worden zodra ‘metAanhangWagen’ aanwezig is.

Figuur 4 schetst een toepassing op Baans erp-suite. Een dergelijke aanpak zal verder geïntegreerd moeten worden met het hele softwareproductieproces (Brinkkemper, 2000). Wat opvalt is dat het zelfs al in deze kleine voorbeelden nuttig is om een onderscheid te maken tussen beperkingen tussen de elementen van één featurediagram en beperkingen tussen elementen van verschillende featurediagrammen. Andere toepassingen van featurediagrammen op softwaresystemen zijn te vinden in Van Deursen & Klint (2002) en Van Deursen e.a. (2002).



Figuur 4. Featurediagram voor een deel van Baans erp-suite

We onderzoeken in dit project een aantal uitbreidingen van featurediagrammen. De taal voor het formuleren van relaties en beperkingen zal uitgebreid moeten worden om alle informatie te kunnen uitdrukken die nodig is voor een softwarekennisbank. Dit betreft niet alleen de aard van en het aantal logische primitieven, maar ook het toevoegen van numerieke waarden om beperkingen als ‘vermogen > 100’ uit te kunnen drukken.
Om effectief te kunnen rekenen met featurediagrammen en de daaraan opgelegde beperkingen, is het nodig om technieken afkomstig uit de modelchecking (zoals *binary decision diagrams*: Bryant, 1992) verder te ontwikkelen. Deze ontwikkeling is om twee redenen van belang: in Van Deursen & Klint (2002) hebben we laten zien dat de oplossingsruimte van featurediagrammen exponentieel groeit, standaard binary decision diagrams-technieken bevatten geen numerieke bewerkingen.
Ten slotte onderzoeken we querymechanismen om de kennis die in featurediagrammen is vastgelegd, te kunnen exploreren en om te zetten in concrete configuraties die geschikt zijn voor distributie.

Webgebaseerde distributie

Relevante protocollen voor de distributie van software via het web zijn WebDAV en DeltaV (James Whitehead, 2002). Deze moeten wellicht aangepast of uitgebreid worden om aan de eisen van Deliver te kunnen voldoen.
Voor de implementatie van deze protocollen in relatie tot de softwarekennisbank zullen we de ToolBus-coördinatiearchitectuur gebruiken (Bergstra & Klint, 1998). De ToolBus is een taalonafhankelijke architectuur voor het combineren van heterogene, gedistribueerde componenten. Het is gebaseerd op een *scripting*taal die distributie en parallellisme ondersteunt. Hierdoor is de ToolBus een goed uitgangspunt voor webgebaseerde software waarin de intelligente softwarekennisbank als centrale server samenwerkt met clients die de

informatie over lokale softwareconfiguraties bij klanten administreren. Gezien de complexiteit van deze interacties zullen hiervoor speciale protocollen ontwikkeld moeten worden.

Conclusies

Zoals uit bovenstaande bespreking blijkt, ontwikkelen featurediagrammen en scripts voor webgebaseerde distributie van software zich langzaam maar zeker in de richting van een domeinspecifieke taal. We zullen onze ervaring daarmee (Klint, 1993; Van den Brand e.a., 2001) gebruiken om voor de uiteindelijke configuratiebeschrijvingstaal interactieve gereedschappen te bouwen zoals syntaxgestuurde *editors*, *type checkers* en hulpmiddelen voor analyse.

Deze korte schets geeft een indruk van de onderzoeksvragen die wij in het Deliver-project bestuderen. Hoewel wij hierbij al met industriële partners samenwerken, houden wij ons zeker aanbevolen voor praktijkvragen die in het kader van dit onderzoek mogelijk opgelost kunnen worden.

Noot

1. Volgens de conventie beginnen atomaire features met een kleine letter en samengestelde features met een hoofdletter.

Literatuur

Bergstra, J.A., & P. Klint (1998). The discrete time ToolBus – a software coordination architecture. *Science of Computer Programming, 31(2-3):205-229, July 1998*.

Brand, M.G.J. van den, A. van Deursen, J. Heering, H.A. de Jong, M. de Jonge, T. Kuipers, P. Klint, L. Moonen, P.A. Olivier, J. Scheerder, J.J. Vinju, E. Visser & J. Visser. The ASF+SDF Meta-Environment: a Component-Based Language Development Environment. In R. Wilhelm (editor). *Compiler Construction (CC '01), volume 2027 of Lecture Notes in Computer Science*, p. 365-370. Springer-Verlag, 2001.

Brinkkemper, S. (2000). Method engineering with web-enabled methods. In S. Brinkkemper, E. Lindencrona, & A. Slvberg (eds.). *Information Systems Engineering: State of the Art and Research Themes*, p. 123-133. Springer Verlag, 2000.

Bryant, R.E. (1992). Symbolic Boolean manipulation with ordered binary-decision diagrams. *ACM Computing Surveys, 24(3):293-318, September 1992*.

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

Deursen, A. van, & P. Klint (1998). Little languages: Little maintenance. *Journal of Software Maintenance, 10:75-92, 1998*.

Deursen, A. van, & P. Klint (2002). Domain-specific language design requires feature descriptions. *Journal of Computing and Information Technology, 10(1):1-18*.

Deursen, A. van, M. de Jonge, & T. Kuipers (2002). Feature-based product line instantiation using source-level packages. In *Proceedings: Second Software Product Line Conference (SPLC2), LNCS*. Springer-Verlag, 2002

Deursen, A. van, P. Klint, J. Visser (2000). Domain-specific languages: An annotated bibliography. *ACM SIGPLAN Notice, 35(6):26-36, June 2000*.

James Whitehead, Jr. E. (2001). WebDAV and DeltaV: collaborative authoring, versioning, and configuration management for the web. In *Proceedings of the twelfth ACM conference on Hypertext and Hypermedia, pages 259-260*. ACM Press, 2001.

Kang, K.C., S.G. Cohen, J.A. Hess, W.E. Novak & A.S. Peterson (1990). *Featureoriented domain analysis (FODA) feasibility study. Technical Report CMU/SEI-90-TR-21*. Software Engineering Institute, Carnegie Mellon University.

Klint, P. (1993). A meta-environment for generating programming environments. *ACM Transactions on Software Engineering and Methodology, 2(2):176-201, April 1993*.

OESO (2001). The software sector: Growth, structure and policy issues. *Technical Report DSTI/ICCP/IE(2000)8/REV2. OESO, October 2001*.

Gerco Ballintijn
is postdoc bij de onderzoeksgroep Interactive Software Development and Renovation van het CWI. E-mail: Gerco.Ballintijn@cwi.nl.

Sjaak Brinkkemper
is hoogleraar Informatiekunde bij het Instituut voor Informatica en Informatiekunde van de Universiteit Utrecht. E-mail: S.Brinkkemper@cs.uu.nl.

Remy Jansen
is aio bij de afdeling Interactieve Software Development and Renovation van het CWI. E-mail: R.L.jansen@cwi.nl.

Paul Klint
is is hoofd van de afdeling Software Engineering van het CWI en themaleider van de onderzoeksgroep Interactive Software Development en Renovation. E-mail: Paul.Klint@cwi.nl.

Tijs van der Storm
is aio bij de afdeling Interactieve Software Development and Renovation van het CWI. E-mail: T.van.der.Storm@cwi.nl.