Deliver: Intelligent beheer en distributie van software

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

11 augustus 2003

Samenvatting

Het Jacquard project Deliver is 1 juli 2003 van start gegaan en richt zich op het ontwikkelen van een intelligente softwarekennisbank om het configureren en distribueren van productsoftware te verbeteren. Dit artikel geeft een kort overzicht van het voorgenomen onderzoek.

1 Inleiding

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 platformen 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 wetenschappelijk 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.

2 Achtergrond

Het economisch 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 [12]. In 1999 werd de totale markt voor productsoftware geschat op 155 miljard dollar, met een jaarlijks 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.

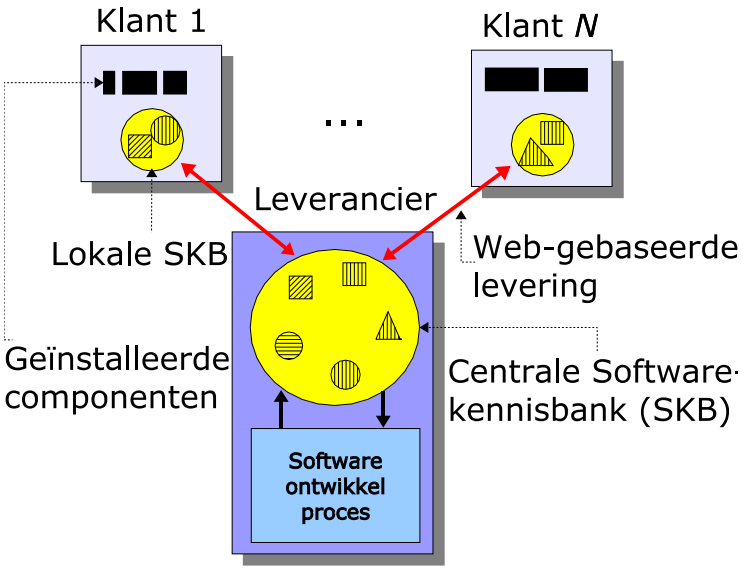
Productsoftware kan onderverdeeld worden in een aantal categorieën. De belangrijkste zijn *consumententoepassingen*, *softwareontwikkelgereedschappen*, *systeemsoftware* en *bedrijfstoepassingen*. We gaan hier alleen op consumenten- en bedrijfstoepassingen in.

Consumententenproducten 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.

Het is voor productsoftware daarom essentieel om voor elke klant nauwkeurig bij te houden welke versies van componenten in gebruik zijn. Om in staat te zijn om deze verschillende versies van componenten ook feitelijk te kunnen leveren, vereist dit op zijn beurt een nauwkeurige administratie van alle *sofware-artefacten* 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, data (iconen, geluids- en videoafragmenten) en dergelijke. 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 ten aanzien van de beschikbaarheid, stabiliteit en prestaties van andere componenten. Als we ook nog het aantal configuratieparameters op verschillende platforms in beschouwing nemen (hardware, operating system, databasesysteem, gebruikerinterface, middleware, enz.) 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/hardware platforms 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 nodig voor deze component zijn geïnstalleerd bij de klant?



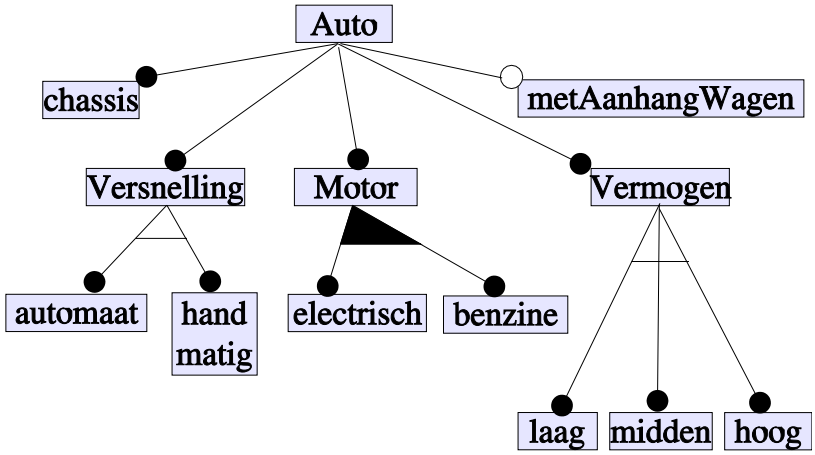
Figuur 1: Globale Architectuur

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.

Om hier verandering in te brengen is het nodig om het proces van bouwen, distribueren en integreren meer inzichtelijk en beheersbaar te maken. Dit kan alsvolgt bereikt worden (zie Figuur 1):

- Een *Intelligente Softwarekennisbank* (ISKB) die uitputtende informatie bevat over alle software-artefacten en hun onderlinge verbanden en beperkingen. Bovendien bevat de kennisbank regels omtrent 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* voor het distribueren, opwaarderen en vervangen van software componenten. Met de informatie die de softwarekennisbank genereert kan de klantconfiguratie op afstand beheerd en aangepast worden.



Figuur 2: Featurediagram voor Auto.

3 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. We gaan nu op beide aspecten nader in.

3.1 Kennisrepresentatie

Als uitgangspunt voor de softwarekennisbank gebruiken we *featurediagrammen* [5, 8] voor het representeren van de hierarchische structuur en variabiliteit van de softwarecomponenten van de leverancier.

Figuur 2 geeft een featurediagram voor een eenvoudige auto en is ontleend aan [5]. 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 gesloten 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 gedefineerd 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”): hetzij `automatisch` hetzij `handmatig` kan gekozen worden worden maar niet allebei tegelijk.

De `Motor` kan `electrisch` zijn of op `benzine` lopen. De gesloten driehoek geeft aan dat het een niet-exclusieve keuze is (“”more-of”): of `electrisch`, of `benzine`, of beiden zijn mogelijk.

[1] We gebruiken de conventie dat atomaire features met een kleine letter beginnen en samengestelde features met een hoofdletter.

```
Auto: all( chassis, Versnelling, Motor, Vermogen, metAanhangWagen? )

Transmissie: one-of( automatisch, handmatig )

Motor: more-of( electrisch, benzine )

Vermogen: one-of(laag, midden, hoog)
```

Figuur 3: FDL versie van het featurediagram voor Auto.

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 [8] een equivalente tekstuele versie voorgesteld met de naam Feature Description Language (FDL). Een FDL versie van de auto is te zien in Figuur 3.

Een belangrijk onderdeel van FDL zijn de *beperkingen* die opgelegd kunnen worden aan een selectie van features. Een typische beperking in het autovoorbeeld zou kunnen zijn `metAanhangWagen requires hoogVermogen`. Dit heeft als effect dat alleen de `hoogVermogen` variant geselecteerd kan worden zodra `metAanhangWagen` present is.

Figuur 4 schetst een toepassing op Baans ERP suite. Een dergelijke aanpak zal verder geïntegreerd moeten worden met het hele softwareproductieproces [3]. 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 [8, 6].

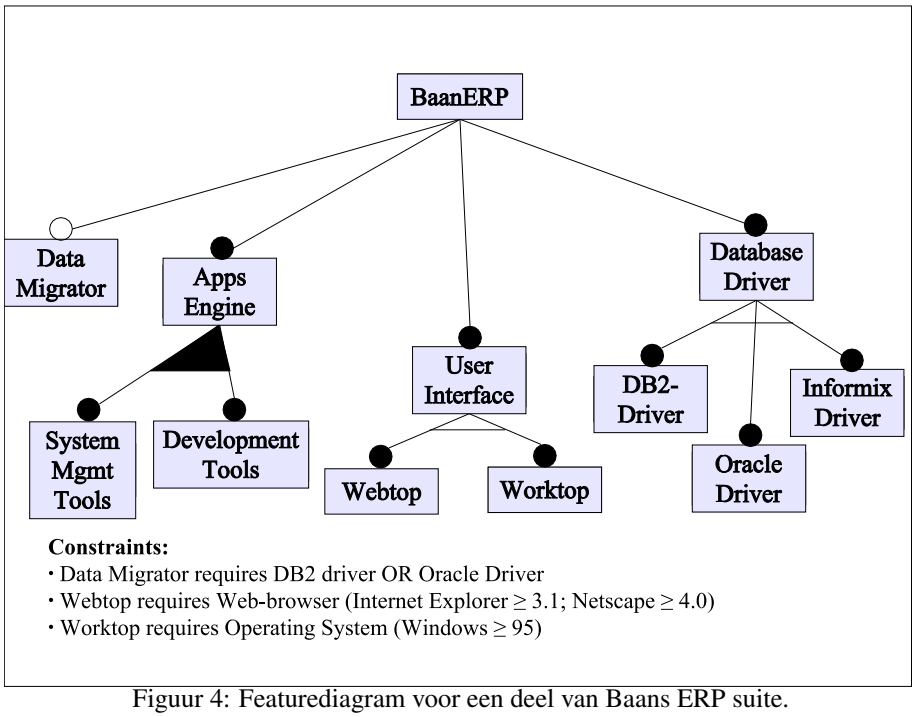
We zullen in dit project de volgende uitbreidingen van featurediagrammen onderzoeken:

- 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 [4]) verder te ontwikkelen. Deze ontwikkeling is om twee redenen van belang: (*i*) in [8] hebben we laten zien dat de oplossingsruimte van featurediagrammen exponentieel groeit, (*ii*) standaard BDD technieken bevatten geen numerieke bewerkingen.

3.2 Webgebaseerde distributie

Relevante protocollen voor de distributie van software via het Web zijn WebDAV en DeltaV [10]. Deze moeten wellicht aangepast of uitgebreid worden om aan de eisen



Figuur 4: Featurediagram voor een deel van Baans ERP suite.

van Deliver te kunnen voldoen.

Voor de implementatie van deze protocollen in relatie tot de softwarekennisbank zullen we de ToolBus coördinatie-architectuur gebruiken [1]. De ToolBus is een taalonafhankelijke architectuur voor het combineren van heterogene, gedistribueerde componenten. Het is gebaseerd op een scriptingtaal 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.

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 (DSL) [7, 9]. We zullen onze ervaring daarmee [11, 2] gebruiken om voor de uiteindelijke configuratiebeschrijvingstaal interactieve gereedschappen te bouwen zoals syntaxgestuurde editors, type checkers, en hulpmiddelen voor analyse.

4 Conclusies

Deze korte schets geeft een indruk van de onderzoeksvragen die wij in het Deliverproject zullen 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.

Referenties

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

[2] M.G.J. van den Brand, 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, and 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*, pages 365–370. Springer-Verlag, 2001.

[3] S. Brinkkemper. Method engineering with web-enabled methods. In S. Brinkkemper, E. Lindencrona, and A. Slvberg, editors, *Information Systems Engineering: State of the Art and Research Themes*, pages 123–133. Springer Verlag, 2000.

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

[5] K. Czarnecki and U. Eisenecker. *Generative Programming: Methods, Techniques and Applications*. Addison-Wesley, 2000.

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

[7] A. van Deursen and P. Klint. Little languages: Little maintenance. *Journal of Software Maintenance*, 10:75–92, 1998.

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

[9] A. van Deursen, P. Klint, and J. Visser. *Domain-specific Languages*, volume 28, pages 53–68. Marcel Dekker, Inc. New York, 2002.

[10] Jr. E. James Whitehead. 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.

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

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