thema

De afgelopen JavaOne conferentie in San Francisco was een interessante paradox waar te nemen. Java moet binnen nu en twee jaar naar een developer community van tien miljoen developers uitgroeien, zo werd gesteld. Tegelijkertijd was er de bezorgdheid van veel leveranciers en ontwikkelaars over de toegenomen complexiteit in Java, met name in J2EE. Op het eerste gezicht lijken dit twee conflicterende zaken. Het toegankelijk maken van Java voor een groter publiek is immers niet gebaat bij een steeds complexer wordende omgeving.

Metadata: nieuwe feature in JDK 1.5

Program Annotation Facility voor Java

Het credo voor de komende releases van zowel J2SE als J2EE zal dan ook "Ease-of-Development" (EoD) zijn: het toegankelijker en minder complex maken van Java. Eén van de speerpunten van EoD wordt gevormd door de introductie van "JSR-175 Program Annotation Facility for the Java Programming Language". Momenteel heeft de JCP hiervan de specificatie als public draft beschikbaar op haar website. Reden genoeg dus om er eens een eerste blik op te werpen.

METADATA De Program Annotation Facility voor Java, liefkozend door velen als "metadata" bestempeld, is één van de spraakmakende nieuwe features van JDK 1.5 (Tiger), die gepland staat voor de zomer van 2004. Met behulp van metadata is het mogelijk om zogenaamde annotaties (bijschriften) te plaatsen bij velden, methodes en classes in Java broncode. Het is tevens mogelijk om zelf de implementatie van zo'n (custom) annotatie te realiseren. Dit mechanisme vertoont enige gelijkenis met bijvoorbeeld de custom tag library's afkomstig uit de JSP en web development wereld. De in de broncode opgenomen annotaties veranderen uit zichzelf niets aan de semantiek van een programma, maar ontwikkel- en/of deployment tools zullen deze annotaties interpreteren en daar op een bepaalde manier mee omgaan. Dit zou bijvoorbeeld kunnen resulteren in het uitgenereren van additionele broncode, XML configuratie documenten (deployment descriptor) of andere zaken.

ASPECTEN VAN ANNOTATIONS Hoewel ik persoonlijk mijn twijfels heb over het mengen van programmacode en deployment informatie, zit er meer achter JSR-175 dan alleen maar het overbodig maken van een deployment descriptor. In wezen is het een eerste afgeleide van iets dat Aspect Oriented Programming (AOP) wordt genoemd. Aspect Oriented Programming is een filosofie waarin als aanvulling op Object Oriented Programming bepaalde aspecten aan de programmacode worden toegevoegd. Een aspect kan simpelweg worden vertaald naar een bepaalde invalshoek van waaruit naar een stuk broncode kan worden gekeken. Vanuit die invalshoek zou men kunnen redeneren dat een stuk broncode iets doet met bijvoorbeeld het aspect "distributie". Over dit aspect kan bij de code een annotatie worden gemaakt, waarin de eisen die de code stelt aan distributie kunnen worden verwoord in een formeel gedefinieerde taal. Deze annotatie kan vervolgens worden geïnterpreteerd door een tool die op basis van de toegespeelde informatie de juiste acties onderneemt om dit stuk code in staat te stellen om daadwerkelijk gedistribueerd te zijn. Zo kan bijvoorbeeld automatisch een stub worden gegenereerd, waarmee remote procedure calls naar een instantie van deze class kunnen worden gedaan.

TOEPASSINGEN Zoals gezegd zijn de mogelijkheden van een annotatiemechanisme legio, aangezien het voor veel zaken direct toepasbaar is. Denk bijvoorbeeld aan het gebruik van annotaties voor de bekende @deprecated tag van JavaDoc, of als vervanger voor de zogenaamde "marker interfaces" (interfaces zonder inhoud, maar puur als typering) zoals bijvoorbeeld java.io.Serializable of java.rmi.Remote. De genoemde

toepassingen zijn voor het grootste gedeelte bedoeld voor ontwikkeltools, die compile-time reageren op de aangetroffen annotaties. Tijdens runtime hoeft deze informatie dus niet (meer) beschikbaar te zijn, waardoor de class file niet onnodig groot hoeft te worden, wat weer bespaart in geheugengebruik en niet onnodig ten koste gaat van performance. Er is wel een mogelijkheid om runtime annotatie types te maken, die door de compiler als class file attributen worden opgeslagen. Via de (nieuwe) reflection API is het mogelijk om deze annotaties tijdens runtime te inspecteren (op de gecompileerde class). Dit laatste heeft wel een aanpassing in de Java Virtual Machine Specification tot gevolg. Hiervoor wordt het Class File Format uitgebreid met vijf nieuwe attributen die de annotatie op de class file beschrijven.

IMPLEMENTATIE De implementatie van een annotatie kan worden gezien als een interface declaratie, zij het een zeer speciale vorm daarvan. Annotatie type declaraties zijn geldig waar interface declaraties geldig zijn en hebben dezelfde scope en toegankelijkheid. Om een annotatie en een interface uit elkaar te kunnen houden is voor de annotatie gekozen om het keyword "@interface" te gebruiken. Er is heel bewust gekozen om gebruik te maken van twee reeds gereserveerde keywords ("@" en "interface") in plaats van het introduceren van een nieuw keyword. Stel dat was gekozen voor een keyword als "annot", "annotation" of "metadata", dan is de kans groot dat heel veel bestaande software moet worden aangepast omdat deze namen daar misschien als variabele naam zouden kunnen bestaan. Deze code zou met een JDK 1.5 compiler dan niet meer compileren. Het gebruik van de "@" kent nog twee anekdoten; de gelijkenis die het vertoont met JavaDoc en de uitspraak van "@", namelijk "at", wat op zich weer als afkorting kan worden gelezen voor "annotation type".

VOORBEELDEN Java programmeurs zullen op twee manieren met annotaties in aanraking komen: of ze maken nieuwe annotatie types, of ze gebruiken ze. Een voorbeeld van het maken van simpele annotaties:

```
// Voorbeeld single-member annotation type
declaratie
public @interface Copyright {
    String value();
}
```

Deze is als volgt te gebruiken:

```
// Gebruik van single-member annotation
@Copyright("2004 Info Support BV, All rights
reserved.")
public class ProfessionalDevelopmentCenter {
... }
```

Als je eenmaal de smaak te pakken hebt, dan kun je bijvoorbeeld ook complexere annotaties maken:

```
// Voorbeeld complex annotation type declara-
tie
public @interface Name {
    String first();
    String last();
}
public @interface Author {
    Name value();
}
```

In het gebruik zien deze er als volgt uit:

```
// Gebruik van complex annotation
@Author(@Name(first = "Bert", last =
"Ertman"))
public class MyClass { ... }
```

Naast de gepresenteerde voorbeelden zijn er nog allerlei geavanceerde toepassingen, zoals het annoteren van annotaties (meta-annotaties genaamd) en het combineren van annotaties met Enumerated Types of Generics (twee andere, interessante, nieuwe, features in J2SE 1.5)!

In eerste instantie zullen J2SE API's worden uitgebreid met eenvoudige, bruikbare annotatie types, bijvoorbeeld als alternatief voor marker interfaces. Het wachten is op de aankondiging van een zogenaamde Standard Annotation Type Library, ter versimpeling of automatisering van bepaalde zaken zoals bijvoorbeeld het open source framework Xdoclet daar nu een oplossing voor biedt. Met het oog op de voorgenomen versimpeling van de huidige J2EE technologie zou de declaratie van een Enterprise JavaBean er dan bijvoorbeeld als volgt uit kunnen zien:

```
public class @session MyBean {

@remote @transaction("required")
public void helloWorld() { ... }

...
}
```

Zoals is te zien in het voorbeeld kan het vervaardigen van Enterprise Beans met behulp van annotaties en afgesproken default instellingen worden versimpeld tot het declareren van een eenvoudige Java class, zonder het expliciet zelf hoeven maken van allerlei interfaces of deployment descriptors. De Integrated Development

Environment (IDE) zal de annotaties gebruiken om tijdens compile of deployment tijd de juiste acties te ondernemen.

AFHANKELIJKHEDEN Belangrijk om te beseffen is dat er een afhankelijkheid bestaat tussen de source code waarin annotaties worden gebruikt en de implementatie van de annotaties zelf. Deze afhankelijkheid is niet anders dan normale classes of interfaces met elkaar hebben. De source code zal dus ook import statements moeten bevatten voor alle gebruikte annotatie types en om te kunnen compileren zullen de annotatie types dus ook daadwerkelijk beschikbaar moeten zijn. Op zich is dit gedrag natuurlijk gewenst, immers de annotaties nemen ons werk uit handen, zoals het genereren van source code of configuratie files, dus zijn ze onmisbaar. Er zijn echter ook annotaties die een meer beschrijvende functie hebben, zoals bijvoorbeeld het eerder beschreven voorbeeld van de "copyright" annotatie. In dit laatste geval zal de code prima functioneren, ook al mist de implementatie van deze annotatie bij het compileren. In de specificatie is nog geen mogelijkheid opgenomen om bij een annotatie type aan te geven in hoeverre deze bij het compileren of tijdens runtime verplicht aanwezig moet zijn. De komende maanden zal JSR-175 verder worden uitgewerkt om mede op basis van de terugkoppeling uit de community onder andere dit soort issues op te lossen.

CONCLUSIE Het gebruik van metadata gaat ongetwijfeld bijdragen aan de versimpeling van Java. Een eerste kennismaking met metadata zal in de vorm zijn als vervanger van marker interfaces of in plaats van bepaalde JavaDoc functionaliteit. Echter, naar mate men meer bekend zal raken met de kracht van een dergelijk fenomeen zal het al gauw de motor worden voor het verbergen van veel van de complexe mechanismen. Bijvoorbeeld in J2EE. Voor een doorgewinterde Java programmeur wordt het wel even wennen om in plaats van "implements java.io.Serializable" nu ineens @serializable te gebruiken in de class declaratie, echter binnen afzienbare tijd wil hij niets anders meer. Omdat source code die gebruik maakt van annotaties dezelfde afhankelijkheden heeft met annotatie implementatie classes als met willekeurige andere classes, is het aan te raden zoveel mogelijk gebruik te maken van tot standaard verheven annotaties in plaats van eigen proprietary annotaties. Op het moment van schrijven is nog niets bekend van een dergelijke verzameling van standaard annotaties, maar reken er maar op dat deze snel gaan komen. Tenslotte, bij de introductie van J2SE 1.5 zullen annotaties voornamelijk als nieuwe feature worden gepresenteerd, maar zodra in de loop van datzelfde jaar de volgende versie van J2EE uitkomt, zullen ze een wezenlijk onderdeel gaan vormen van de wijze waarop in de nabije toekomst Java applicaties worden ontwikkeld.

Ing. Bert Ertman is als IT-architect werkzaam bij Info Support B.V. te Veenendaal en sinds 1996 actief op het vlak van Java development. Wilt u reageren op dit artikel? Mail naar berte@infosupport.com.

PATCHES Patches PATCHES Patches PATCHES Patches PATCHES

IBM investeert weer een miljard dollar

In december 2000 maakte IBM bekend een miljard dollar te investeren in Linux. Die bekendmaking, evenals het bedrag, heeft zeker een katalyserende werking op de Linux-groei gehad. Iets meer dan 3 jaar later maakt IBM iets soortgelijks bekend, maar dan met betrekking tot Websphere.

Daaraan voorafgaand waren er twee kleine schermutselingen tussen Sun en IBM: op 21 januari schreef Jonathan Schwartz, EVP van Sun een open brief aan IBM's CEO Samuel J. Palmisano. IBM wilde overgaan tot het gebruiken van Linux op de desktop en Schwartz vertelde hem hoe eenvoudig zulks was met behulp van Sun's Java desktop. Palmisado op zijn beurt schreef een open brief aan Sun, waarin het bedrijf opgeroepen werd om Java aan de open source gemeenschap over te doen.

Op 2 maart tenslotte, maakte Palmisano bekend een miljard dollar te willen besteden om ontwikkelaars over de gehele wereld voor Websphere te winnen. Volgens een bericht in de Financial Times wil IBM proberen ontwikkelaars te verleiden met co-marketing overeenkomsten, gratis software, goedkope ontwikkeltools en toegang tot grote klanten accounts op de voorwaarde dat ze hun software toegankelijk zouden maken voor de IBM platvormen, met name voor WebSphere. Volgens de Financial Times zou het IBM vooral gaan om het overhalen van .Net ontwikkelaars. Onderdeel van deze strategie zou een uitbreiding naar verticale markten zijn.

Volgens een persbericht van IBM levert het bedrijf een combinatie van Express producten, zodat het voor partners mogelijk zou zijn MKB-oplossingen te maken die zowel uniek als reproduceerbaar zijn.Bedrijfsmatig middleware deployment zou minder complex en goedkoper moeten worden. IBM's Integrated Runtime, gepreconfigureerde middleware waarop gemakkelijk software te schrijven, installeren en te deployen is voor eindgebruikers, maakt gebruik van IBM's Express portfolio, inclusief WebSphere Application Server Express en DB2 UDB Express.

Omdat Integrated Runtime op modulaire componenten is gebaseerd, kunnen ISV's de deployment aanpassen aan de bestaande technologie van de klant, maar ook de IT-infrastructuur gemakkelijk schalen. Ook is er een via IBM's developerWorks site leverbaar Migration Station dat het Microsoft-geöriënteerden en anderen mogelijk maakt gemakkelijk te migreren naar een IBM-omgeving.