1. CrossMarx

Het winnende team...

De winnaar van Development Tools 2006 heeft ons voor heel wat hoofdbrekens geplaatst. Wat is CrossMarx nu eigenlijk? Is het een Java ontwikkeltool? Een framework? Is het een 4 GL? Een code-generator? Zelf noemen ze het een application engine, Rick van der Lans heeft besloten het een Java 4GL te noemen, en daar laten we het voor nu even bij.

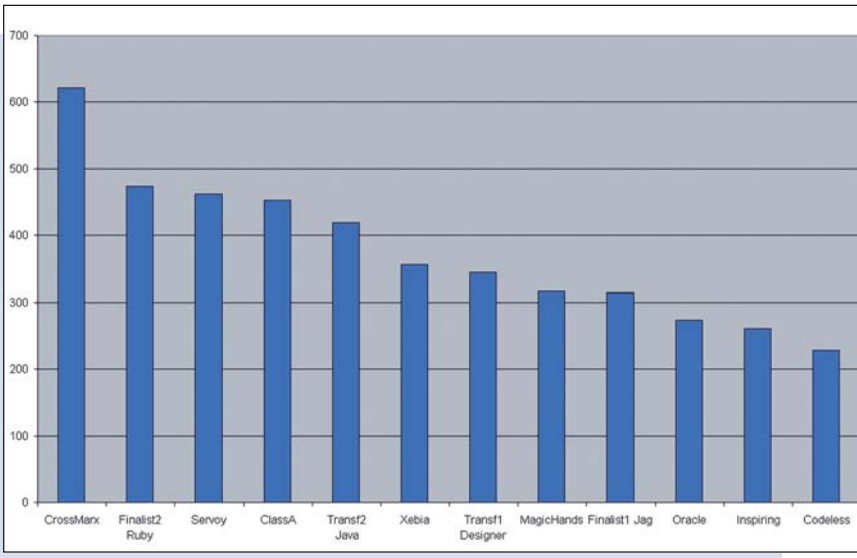
Het team is heel wat duidelijker te categoriseren; het zijn de ontwikkelaars van de tool. Zelf zeggen ze van hun product: 'In het algemeen geldt dat de Application Engine optimaal past bij mensen die de wil hebben om zelf hun maatwerkapplicaties te realiseren. Ze dienen in staat te zijn om processen en gegevens binnen bedrijven te analyseren en te beschrijven. Ze hoeven echter niet technisch onderlegd te zijn om een applicatie in een programmeertaal te realiseren. Hun hart moet liggen bij de bedrijfsprocessen, en niet bij de techniek. Echter enige technische kennis is wel nodig, maar die hoeft het niveau van algemene computervaardigheden niet te overstijgen.'

Het tool maakt geen gebruik van modelleertechnieken. De makers zeggen wel dat mogelijk zou moeten zijn om aan te sluiten bij ene business modelleertool. Het valt vooral op door de keurige OO-manier waarop alles is opgebouwd. Veel kan gedaan worden met standaardknoppen die functies aanroepen die weer uit te breiden zijn. Daardoor ontstaan nieuwe Java-klassen die er op zich weer heel netjes en overzichtelijk uitzien. Het tool is duidelijk database-gebaseerd. Uit de waarden van velden kunnen andere waarden worden afgeleid door middel van standaardoperaties met booleans, rekenoperaties, concatenaties et cetera.

De code wordt niet gegenereerd. In principe is het mogelijk met Java-code nieuwe componenten te bouwen, die weer in het framework opgenomen worden. Tijdens de race hebben de deelnemers dat ook gedaan, maar naar eigen zeggen waren ze heel tevreden over het feit dat ze dat niet vaak hebben hoeven te doen. In die zin stelde de opgave wel bijzondere eisen: 'Voor de helft van onze klanten hebben we nog nooit code hoeven te schrijven.' Dat deze manier van werken echt productief is, blijkt natuurlijk uit het feit dat dit team verreweg de meeste functionaliteit gebouwd heeft in de twee dagen. Maar ook tijdens de demonstratie na de wedstrijd wist het team te laten zien hoe eenvoudig een functie als het toevoegen van het versturen van e-mails binnen een scherm toe te voegen zijn. In de toekomst de makers van het tool van de componenten open source maken, zodat anderen daar weer eigen componenten aan kunnen toevoegen. Het tool bewaart de code in een repository, waarvoor een relationele database gebruikt wordt, in dit geval MySQL. Een nadeel van het tool is, dat de gebruikersinterface nog vrij Spartaans is. Uit de vormgeving spreekt niet de eenvoud die het tool in feite kenmerkt.

Desgevraagd gaf CrossMarx toe dat ze daar ook snel aan willen gaan werken. Gebruikers met wie ik gesproken heb, stoorden zich er echter niet zo aan. Rick van der Lans noemt het een Java-4GL. (zie ook het eerste deel van dit artikel). Daarbij valt op te merken, dat het

tool tegelijk wel en niet open is. Het werkt gewoon met Java, en kan met standaard applicatieservers overweg. Het is echter weer niet zo heel open omdat het – op dit moment – heel moeilijk is om het tool voor eigen gebruik aan te passen. Wat ontbreekt er nog aan? Modelleermogelijkheden, zeker voor het opstellen van een datamodel. Dat laatste zag het team ook wel, want het opstellen van het datamodel heeft ze relatief vrij veel tijd gekost. Enige extra aandacht aan de UI van de workbench zelf zou ook geen kwaad kunnen. CrossMarx heeft een duidelijke voorsprong opgebouwd op de andere teams. Persoonlijk denk ik dat dit het gevolg is van het feit dat ze de voordelen van een 4GL wel heel slim met de voordelen van OO hebben gecombineerd: een Java-4GL dus.



De uitslag van Development Tools 2006

2. Ruby

Een van de aangename verrassingen van deze race was de deelname van het Ruby Team van Finalist. Het was eigenlijk ook een verrassing voor het team zelf, dat zich normaal alleen met het schrijven van Java-code bezig houdt. Als Ruby-team waren ze dan ook niet heel erg ervaren, wat ze er niet van weerhield om de tweede plaats in de wacht te slepen.

Eigenlijk vonden ze dat ze al gewonnen hadden door mee te doen, want dat betekende dat ze hun chef Jean-Luc van Hulst al overtuigd hadden van het belang van Ruby. Ruby on Rails (hierna: ROR) wordt bij Finalist nog niet echt ingezet, al zou dat na deze wedstrijd wel eens kunnen veranderen. Het team

```
class Customer < ActiveRecord::Base
  EMAIL_VALIDATION = /^([^@\s]+)@((?:[-a-z0-9]+\.)+[a-z]{2,})|(localhost)$/
  belongs_to :user, :dependent => true
  has_many :registrations
  serialize :registrationsteps

  validates_length_of :cityname, :maximum => 30
  validates_length_of :companyname, :maximum => 30
  validates_length_of :countryname, :maximum => 30
  validates_length_of :email, :maximum => 30
  validates_format_of :email, :with => EMAIL_VALIDATION
  validates_length_of :firstname, :maximum => 30
  validates_length_of :function_description, :maximum => 20
  validates_length_of :lastname, :maximum => 30
  validates_length_of :postcode, :maximum => 6
  validates_format_of :postcode, :with => /^\d{4}[a-z]{2}$/i, :message => "
  validates_length_of :sex, :maximum => 1
  validates_format_of :sex, :with => /^[MF]$/, :message => "moet 'M' of 'F'
  validates_length_of :streetname, :maximum => 30
  validates_length_of :streetnumber, :maximum => 5
  validates_length_of :telnumber, :maximum => 15
  validates_format_of :telnumber, :with => /^\d{3}-\d{7}$/, :message => 'mc
  validates_presence_of :cityname, :companyname, :countryname, :firstname,

  # test postcode tegen bestaande records
  validate do |this|
    that = Customer.find(:first, :conditions => ['postcode = ?', this.postc
    if that && that.cityname.downcase != this.cityname.downcase
      this.errors.add(:postcode, "gegeven postcode hoort bij '#{that.cityn
    end
  end
```

Een voorbeeld van de Ruby-code

vond de opgave 'perfect voor ROR. Als er een klant zou komen met zo'n opgave zou ik zeggen: dat doen we met ROR'. Wat maakte de opgave zo geschikt: 'Dat het overzichtelijk is, niet megacomplex. En alles zit er zo'n beetje in: webservices, webclients, want het is een heel mooie tool om webclients mee te maken, het doet veel met Ajax.' Het Ruby-team was het enige team dat niet terugschrok voor de business rules: 'De business rules waren ook heel gemakkelijk: je hebt heel goed hooks, je kunt overal inprikken. Ook als je achteraf nog complete auditing zou willen hebben dan is dat vrij gemakkelijk te maken zonder dat je bestaande code hoeft aan te passen.' De architectuur bestaat uit MySQL, de development server van ROR waarop gedeployed is, en een appserver geschreven in Ruby. 'In

productie zou je dat waarschijnlijk anders doen, meestal een opstelling met Apache of LighTTP, een heel lichgewicht http server'. Rails (het framework bij Ruby) lijkt veel op de frameworks die teams als Xebia gebruiken, met daarnaast mogelijkheden voor unit testing en functioneel testen. Als Smalltalk-erfgenaam is Ruby natuurlijk heel erg OO. Opvallend is vooral de enorm compacte (en eenvoudige) code.

Een groot voordeel is ook dat er dus geen recompile en opnieuw deployen noodzakelijk is: het framework herkent dat een klasse is veranderd. Aangezien Ruby een van de scripttalen is die in de Mustangversie van Java ondersteund zullen gaan worden, zou het wel eens een grote toekomst tegemoet kunnen gaan.

Bij het team was het in ieder geval was het in ieder geval liefde op het eerste gezicht. Wat het nog een beetje gemist had, waren gerichte functies voor bepaalde opgaven die ze nu – omdat ze er nog niet iedere dag mee werken – nog niet tot hun beschikking hadden. De hoeveelheid plug-ins groei met de dag, maar de plug-in die ze nu voor autorisatie uitgekozen hadden, bleek achteraf zo veel te kunnen dat hij meer tijd gekost heeft dan wanneer ze het geschreven zouden hebben: 'Je schrijft het allemaal zo snel, dat het uitzoeken van hoe zo'n plug-in werkt meer tijd kost.' Hier en daar wreekte zich dus nog het feit dat de ROR-kennis van het team vooral het product was van avondlijk hobbyisme. Te hopen valt dat er binnen Finalist nu tijd vrij gemaakt zal worden om het team ook eens een echt project te laten doen met Ruby.

3. Servoy

Servoy lijkt in een aantal opzichten op de nummers een en twee: het lijkt op Crossmarx in de 4GL-achtige manier waarop je applicaties kunt bouwen en op ROR doordat het eveneens van een scripttaal gebruik maakt, al is het hier dan Javascript. De voornaamste reden dat het tool niet evenveel punten heeft gehaald als ROR is dat de userinterfaces er een stuk minder mooi uitzagen.

Servoy is overigens ooit ontstaan uit een bedrijf dat een zeer populair plug-in voor Filemaker verkocht, en heeft daar ook, nog een aantal grote klanten overgehouden die nu met Servoy werken. Het datamodel heeft het team gemaakt met Sybase PowerDesigner, de ingelezen tabellen werden uitgebreid op basis van de opdracht. Sommige meteen, andere pas later. Stored procedures neemt Servoy overigens niet waar. Door het klikken op tabellen is het heel eenvoudig schermen te maken, de velden worden er door het tool uitgehaald.

Bij deleten van tabellen 'ziet' de code dat en omgekeerd. Zonder dat de gebruiker SQL hoeft te schrijven zijn standaardoperaties als insert, delete, find en dergelijke mogelijk. Er is een query-generator, maar het intypen van SQL is ook mogelijk. De business rule voor het e-mailadres had het team graag als reguliere expressie op Internet opgezocht, maar die mogelijkheid verviel nu, dus die hebben ze zelf moeten schrijven. In die volgende versie kunnen overigens business rules ook globaler gebuikt worden, doordat ze op kolommen gelegd kunnen worden.

Blobs worden standaard als image media field aangemaakt met review-mogelijkheid. Het tool ondersteunt tekst-zoekmogelijkheden (al moest voor de opdracht er wat logica bij geschreven worden), en het is heel gemakkelijk om in de user interface tabs en dergelijke zichtbaar of onzichtbaar te maken. Het is jammer dat het tool standaard geen mooier user interfaces genereert, al zal dat in de praktijk snel aan te passen zijn. Een sterk punt van het tool is weer dat het werkt op basis van Javascript en Javacode produceert. Javascript-ontwikkelaars zijn gemakkelijker te vinden dan Java-ontwikkelaars. De trainingstijd van de ontwikkelaars die met Servoy willen werken zal vrij kort zijn, en ook hierin komt het tool overeen met de nummers een en twee.

4. Class A

De deelname van een VB.Net team vorig jaar zorgde voor veel speculaties over de prestaties van een Microsoft-team vergeleken met die van een Java-team. Xebia schreef zelfs op de Server Side dat daarmee bewezen was dat Java met frameworks een stuk sneller was dan Visual Studio .NET. Dit jaar was het aan Class A om het tegendeel te bewijzen.

Er waren ook nogal wat verschillen met het team van vorig jaar:

Lees verder op pagina 53

VB versus C#, een op het specifieke gebied van de RAD Race relatief onervaren team tegenover een team dat bestond uit mensen die tijdens presentaties vaak genoeg bewezen hebben de .NET-wereld wel heel goed te kennen, en de 'kale' VS. NET versus VS. NET met een framework/plug in (dat laatste zal overigens toch wel het belangrijkste geweest zijn). De resultaten waren dan ook een stuk beter. Class A was zelfs iets beter dan het beste Java-team.

De verschillen waren echter zo klein dat daarmee geen algemene uitspraken te doen zijn. Qua architectuur was de opzet van dit team niet zo heel afwijkend van Java-teams: aan de basis zat SQL Server, daarboven de data access layer geautomatiseerd door een Nederlands tool LLBLGen Pro ('The n-tier generator and O/R mapper') daar weer boven de business logica laag, deels gegenereerd door handgeschreven, en de UI, deels met code behind. Veel logica zat er twee keer in; zowel in de UI als in de applicatie.

Hier werd meteen het docerende karakter van het Class A-team duidelijk: net als het Xebia-team zouden ze waarschijnlijk beter hebben kunnen scoren wanneer ze zich wat meer gericht zouden hebben op het winnen, in plaats van het zo goed mogelijk maken van de opgave. Het resultaat is echter ook nu al heel goed, zeker wanneer we zien van welk soort tools Class A verloren heeft. Het roept echter ook de vraag op hoe een Micrsosoft-team dat met de nieuwste (wat meer scripttaal-achtige) versie van VB. NET zou werken zou scoren. Nu ja, misschien weten we volgende jaar het antwoord op die vraag.

5. Transfer Java

Het Transfer Java-team eindigde op de vierde plaats – direct achter Ruby. Het beste Java-team had zich dan ook goed bediend uit het assortiment van hulpmiddelen dat Oracle Java-ontwikkelaars aanbiedt. Behalve JDeveloper, werd gebruik van gemaakt van ADF en BC4J en JHeadstart, dit alles bij elkaar leverde kennelijk genoeg versnelling op om diverse ander teams te passeren, waaronder het eigen Designer/Forms-team.

De architectuur was vrij duidelijk: onderop de (Oracle-) database, daarboven de ADF (application development framework) Business Components, daarboven ADF UIX en Apache Struts. Volgens het team heeft JHeadstart wel winst opgeleverd, alhoewel minder dan ze verwacht hadden: 'Een probleem in de huidige Java-stack is dat wanneer je het datamodel aanpast, ook in alle lagen daarboven aangepast moeten worden.'. Het team was ook wel gedwongen het datamodel aan te passen, want ze hadden bij het lezen van de opgave iets over het hoofd gezien.

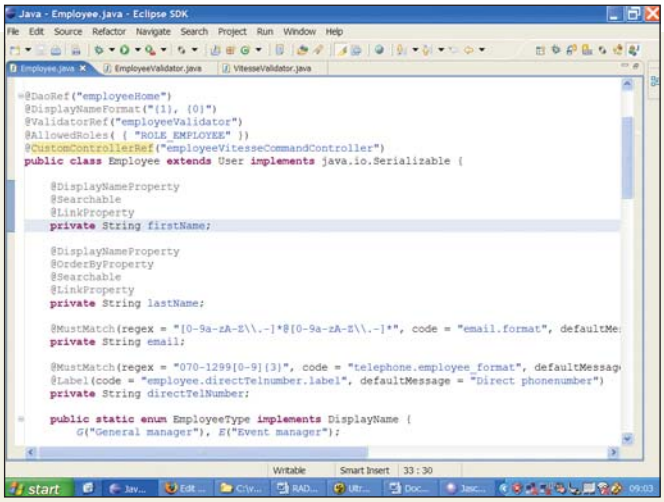
Alle business rules zaten overigens in applicatiemodule. Daarnaast had het team wat problemen met versiebeheer. (Die problemen zouden overigens met de volgende versie van JDeveloper opgelost moeten zijn.) Hoewel het team het beste Java-team was, had het beter kunnen scoren wanneer er – zoals het zelf aangaf – eerder was gaan testen. Niettemin bevestigde het het positieve beeld van de Oracle-Java ontwikkelteams dat ook al was ontstaan op de RAD Race tijdens JavaPolis.

6. Xebia

Het Xebia-team was hetzelfde team dat vorig jaar heel goed scoorde; nauwelijks slechter dan de winnaar. Gezien de aard van de opgave was het eigenlijk vreemd dat het team dit jaar slechts op nummer zes eindigde. Zelf had het team daar ook niet echt een verklaring voor. De architectuur van hun oplossing leek wel op die van vorig jaar, Spring, Hibernate met een eigen framewerkje voor de UI.

Dat laatste was geschreven met de ervaringen van de vorige RAD Race in het achterhoofd. Bijzonder aardig was dat het team gebruik gemaakt had van annotations, een nieuwe mogelijkheid binnen Java. Een echt duide

Het t-shirt van een Xebia-teamlid sprak klare taal



In dit screenshot van de Xebia-code zijn zowel annotations als een reguliere expressie te zien

lijke reden voor de mindere prestatie van het team was er eigenlijk niet. Het maken van het datamodel had het team wel vrij veel tijd gekost, dus je zou je kunnen afvragen of daar geen tijd zou zijn te winnen. Misschien is het ook niet zozeer Xebia dat het slechter gedaan heeft, maar de andere teams die het beter hebben gedaan.

Voor de vele business logica had Xebia niet heel veel hulp van de tools en wanneer een taal als Ruby dan duidelijk compacter is, verklaart dat wel een deel van het verschil. Het team gaf zelf echter misschien ook een deel van een verklaring toen het zei waarom het annotations had gebruikt: het leek ze heel leuk om zo iets nieuws toe te passen.

7. Transfer Forms Designer

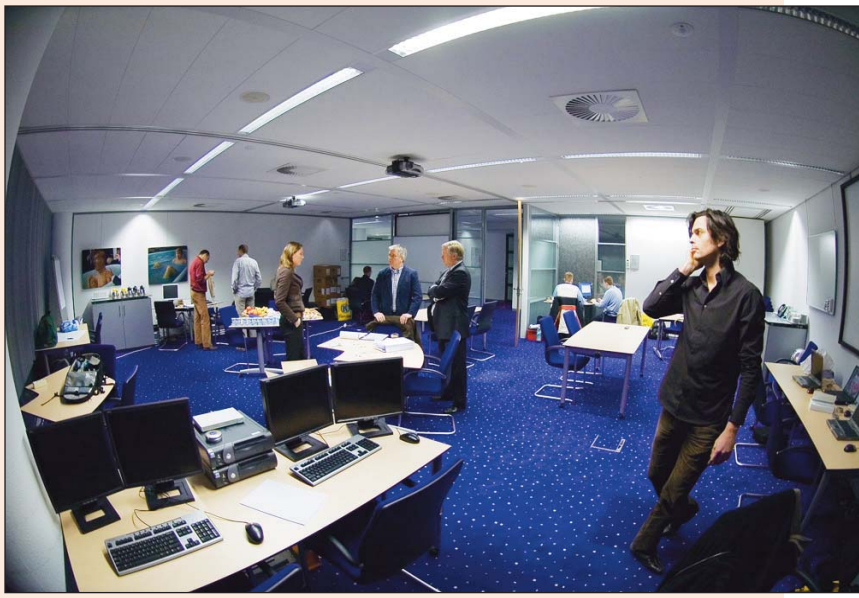
Ook Transfer Solutions deed dit jaar met twee teams mee, die net als bij Finalist twee concurrerende technologieën vertegenwoordigden. Dit Transfer-team werkt – ook al was één van de teamleden slechts 28 jaar – met Designer en Forms, oudere Oracle-technologie die in feite nog gebaseerd is op client/server-applicaties ook al kan deze 10g-versie dan een webclient produceren.

Alhoewel het team zelf dacht dat ze beter gescoord hadden dan hun Java-collega's, bleek het omgekeerde het geval. Waarschijnlijk wreekte zich hier toch het vrij ouderwetse concept van PL/SQL, de taal die ten grondslag lag aan deze tools. In plaats van een mooi klassenmodel met overerving trad bij dit team copy & paste, en dat kostte toch iets meer tijd. Wanneer er - zoals gepland - nog meer wijzigingen zouden zijn gevraagd, zou deze aanpak zich echt gewroken hebben. Checks hebben het team vrij veel tijd gekost, een teamlid (de jongste!) legde alle logica in de user interface, de andere in de database. Constraints werden soms tijdrovend gevonden en hier en daar 'stond een vishaakje de verkeerde kant op', al komen dat soort fouten natuurlijk bij alle teams voor. De meeste checks waren wel zelf geschreven, al waren het oplossingen die het team al eens eerder bedacht had.

Grappig was dat in de code het commentaar stond voor een minder compacte methode dan degene die uiteindelijk geïmplementeerd was (zie afbeelding). Bij het schrijven van een check als stored procedure ontstond een mutating table probleem, waarvoor de workaround toch nog bijna een half uur extra tijd kostte. Een webservice werd in JDeveloper geschreven, bleek echter in Forms niet te werken waarna het team besloot er geen tijd meer aan te besteden. Terugkijkend vond het team wel dat ze het gevoel gekregen hadden dat deze tools langzamerhand wat ouderwets begonnen te worden. Hun conclusie lijkt goed te passen bij de race van dit jaar, waarin deels zeer moderne tools en talen geconfronteerd werden met een opgave die toch ook de eisen van de tijd goed reflecteerde.

(De opgave en meer codevoorbeelden zijn te vinden op www.developmenttools.nl. Voor verdere conclusies zie het redactionele commentaar op pagina 5.)

Foto's en tekst: Dré de Man.



Een paar minuten vóór de aanvang van de wedstrijd