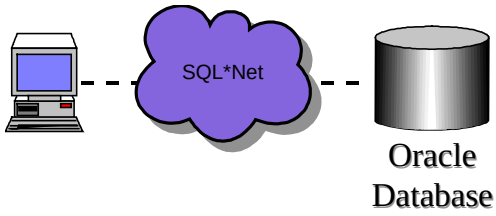
Network Computing Architecture
Toekomstige rol van Oracle Developer marginaal

Door Patrick Koning — *Een aantal jaren geleden realiseerde Oracle zich dat zijn Designer/Developer strategie voor applicatieontwikkeling op termijn plaats zou moeten maken voor een meer open, objectbus-based, multi-tier client/server strategie en ontwikkelde het Network Computing Architecture (NCA).*

De strategie voor applicatieontwikkeling met Designer/Developer valt het beste te typeren met gesloten, database-based en 2-tier client/server.

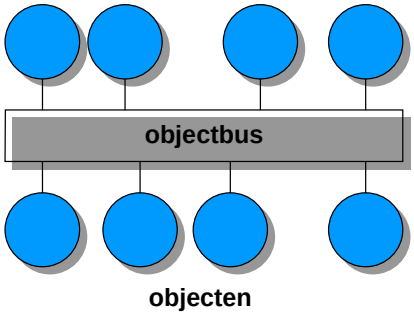


Figuur 1 Designer/Developer applicatieontwikkeling strategie

Middels Designer wordt op analyse & design niveau de applicatie gespecificeerd, waarna men in Developer de uiteindelijke applicatie implemteerd. Een Designer/Developer applicatie bevat aan de client-kant de presentatie in de vorm van Forms en aan de server-kant de gegevens vastgelegd in de Oracle database. De applicatielogica, geschreven in Oracle's eigen PL/SQL-taal, draait of aan de client-kant (fat-client) of aan de server-kant (fat-server). Tussen de client en de server draait Oracle's eigen SQL*Net protocol (zie figuur 1). Een groot probleem van het fat-client model is de duplicatie van proceslogica bij iedere client en de hiermee gepaard gaande distributie en consistentie problemen. Een ander probleem is de belasting van het netwerk door de grote hoeveelheid gegevens die van de server naar de clients getransporteerd moet worden. Het fat server model kent het probleem dat de performance van de applicatie negatief wordt beïnvloed bij een toename van het aantal clients. De server krijgt dan zoveel taken uit te voeren dat deze het resultaat van de taak pas na lange tijd aan de client terug kan geven. Daarnaast geldt dat migratie naar databases van andere leveranciers lastig is omdat de proceslogica middels PL/SQL aan de server kant wordt vastgelegd [1].

Door aan de ene kant het besef van het probleem van de fat-client/fat-server problematiek, de vraag naar open standaarden onder andere om leveranciersonafhankelijkheid te garanderen en de inspanningen van de Object Management Group besloot Oracle een nieuwe strategie te ontwikkelen: het NCA-concept.

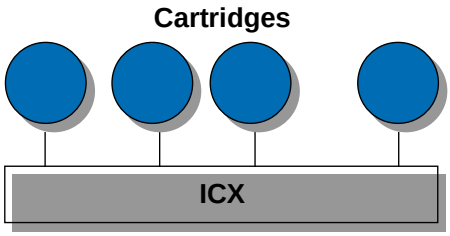
Het NCA-concept
Hét uitgangspunt van het NCA-concept betreft dat applicaties bestaan uit objecten die onderling communiceren middels een object bus (zie figuur 2).



Figuur 2 Objecten communiceren middels een objectbus

Een objectbus biedt de infrastructurele voorziening zodat objecten mogelijk onafhankelijk van hardware, besturingssysteem, programmeertaal en netwerkprotocol met elkaar kunnen communiceren

op basis van open standaarden. Een objectbus biedt dus een transparante vorm van communicatie tussen objecten. De objectbus treedt op als een soort makelaar. Hij redigeert vragen van objecten naar andere objecten en verzorgt de terugkoppeling van het antwoord. De objecten bieden elkaar diensten aan. Juist om de invulling van bovenstaand concept gaan EBM (Everything But Microsoft) en Microsoft [3] de strijd aan. In de visie van Oracle, zoals vastgelegd in het NCA-concept, worden de objecten en de objectbus gerealiseerd middels: Cartridges en ICX (zie figuur 3).



Figuur 3 Cartridges communiceren middels ICX

Cartridges
De cartridges van een applicatie worden volgens het multi-tier client/server architectuurprincipe verdeeld in een drietal typen (zie figuur 4):

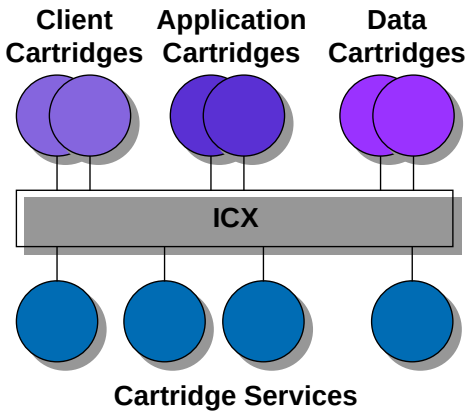
- *client cartridges*
  Een client cartridge verzorgt de presentatie van een applicatie, bijvoorbeeld weergegeven in een webbrowser op een PC of op een Network Computer (NC).
- *application cartridges*
  De application cartridges bevatten de applicatielogica van een applicatie.
- *data cartridges*
  De data cartridges bevatten de gegevens van een applicatie.

De benaming cartridge brengt verwarring met zich mee en is in feite niets meer dan een willekeurig object dat in een Oracle omgeving draait. Zo kan een client cartridge in praktijk gewoon een standaard Java Applet zijn.

Inter Cartridge eXchange
De communicatie tussen de cartridges wordt vormgegeven door de ICX, oftwel Inter Cartridge eXchange. ICX verzorgt de communicatie tussen de cartridges in verschillende fysieke lagen maar eventueel ook tussen cartridges binnen één fysieke laag. De communicatie tussen cartridges in verschillende fysieke lagen is gebaseerd op een drietal standaard protocollen:

- IIOP
  Het Internet Inter ORB Procotol (IIOP) betreft OMG's standaard protocol voor de communicatie tussen verschillende object bussen. Hiermee is het in principe mogelijk om vanuit een cartridge toegang te krijgen tot services die door andere IIOP-compliant object bussen aangeboden worden. En vice versa.
- HTTP
  Het HyperText Transfer Protocol betreft het standaard protocol voor opvragen en uitwisselen van HTML-bestanden.
- SQL*Net
  Oracle's proprietary protocol voor de communicatie tussen clients en Oracle's Universal Data Server.

Doordat de ICX gerealiseerd is middels standaard protocollen is het mogelijk dat een cartridge communiceert met objecten die aan een andere IIOP-compliant object bus hangen en vice versa.



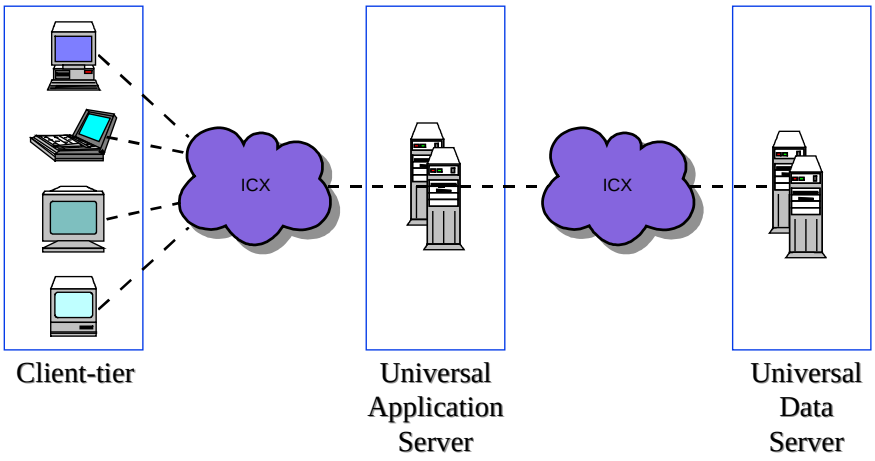
Figuur 4 ICX vormt de basis van het NCA-concept

Naast de verzorging van de communicatie omvat ICX een groot aantal Cartridge Services (zie figuur 4). Deze Services leveren noodzakelijke standaarddiensten voor het controleren van *cartridges* in Oracle--omgeving. De volgende diensten worden (op termijn) geleverd:

- installation — ondersteunt de automatische distributie en installatie van cartridges op de diverse computers in het netwerk.
- registration — ondersteunt het in bedrijf nemen van een cartridge, o.a. het aanmelden van de beschikbaarheid van een cartridge bij de object bus.
- instantiation — ondersteunt het activeren van een cartridge, door de object bus, ter afhandeling van een vraag naar een service van deze cartridge.
- invocation — ondersteunt het op afstand uitvoeren van een bepaalde service geleverd door een cartridge.
- administration — ondersteunt versiebeheer in een gedistribueerde omgeving.
- monitoring — ondersteunt het bekijken van de activiteiten uitgevoerd door de aanwezige cartridges.
- security — voor de beveiliging van het oneigenlijk gebruik van gegevens en resources, o.a. cartridges.
- transactions — voor het definiëren van transacties over cartridges heen.
- messaging and queuing — voor de realisatie van asynchrone communicatie tussen cartridges.
- data access — voor de standaard ontsluiting van gegevens in een database.

Realisatie

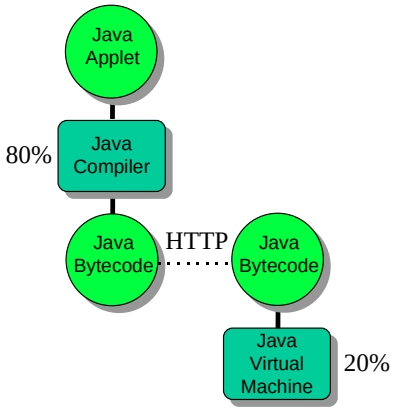
De realisatie van bovenstaand concept heeft Oracle een hoop kopzorgen bezorgd. Allereerst werd het Sedona project opgestart met als doel de realisatie van een nieuwe ontwikkelomgeving voor de invulling van de cartridges en ICX. Na een jaar besloot Oracle in de zomer van 1997 het project stop te zetten met direct gevolg dat men een behoorlijke achterstand op loopt op concurrenten als Microsoft, Netscape, IBM en SUN Microsystems.



Figuur 5 Network Computing Architecture van Oracle

Mogelijke pogingen om Forté, leverancier van een vooraanstaande multi-tier client/server ontwikkelomgeving, op te kopen loopt op niets uit. Uiteindelijk besluit Oracle in zee te gaan met Inprise om het NCA-concept uiteindelijk te realiseren. Op basis van de producten van Inprise levert Oracle een aantal "eigen" producten: JDeveloper, de Universal Application Server en de Universal Data Server.

JDeveloper
JDeveloper 1.0 betreft Oracle's ontwikkelomgeving voor de realisatie van client, application en data cartridges in Java. Java is een krachtig, platformonafhankelijk en object-georiënteerde programmeertaal voor derde generatie internettoepassingen. Met Java worden objecten gecreëerd, zogenaamde Java Applets. Om platformonafhankelijkheid te creëren, vindt compilatie van een Java Applet in twee stappen plaats.



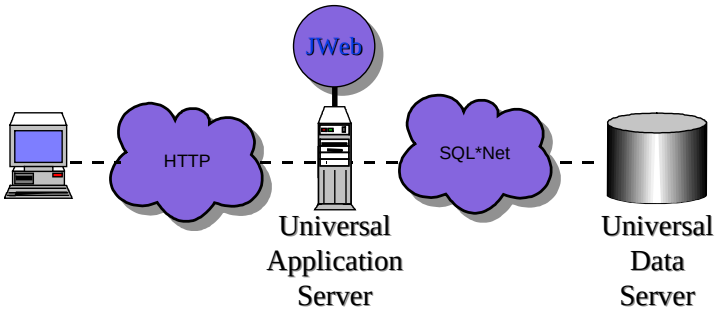
Figuur 6 Java Compiler en JVM

Allereerst wordt de Java Applet gecompileerd tot platformonafhankelijk Java Bytecode (zie figuur 6). Deze Java Bytecode bevindt zich op de webserver. Nadat een webbrowser een verzoek doet voor een Java Applet, verstuurt de webserver de Java Bytecode met behulp van het http-protocol. Vervolgens wordt in de webbrowser de Java Bytecode geïnterpreteerd door de zogenaamde Java Virtual Machine. De platformonafhankelijkheid wordt gecreëerd doordat er diverse implementaties beschikbaar zijn van de Java Virtual Machine.

In de markt wordt deze platform onafhankelijkheid aangeduid met het "Write Once Run Anywhere"-principe (WORA) waarmee gedoeld wordt op het feit dat een applicatie slechts éénmaal gebouwd hoeft te worden en vervolgens op verschillende platformen geëxecuteerd kan worden.

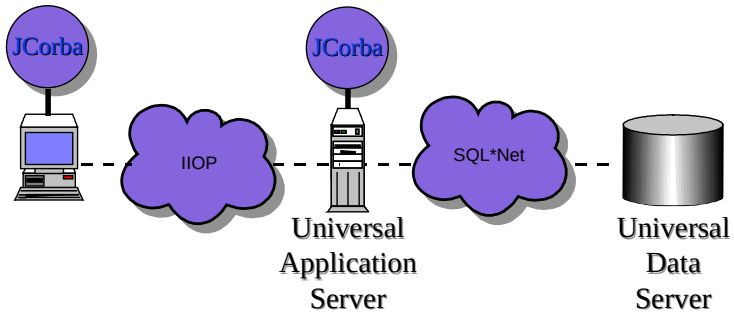
JDeveloper is gebaseerd op JBuilder van Inprise en uitgebreid met een aantal Oracle specifieke zaken voornamelijk op het vlak van de Universal Data Server. Zo zijn er standaard wizards en componenten voor de ontsluiting van de Oracle database middels JDBC. Verder is SQLJ volledig geïntegreerd in de

ontwikkelomgeving. SQLJ betreft de mogelijkheid om SQL-statements en Java-code te mixen. Middels een pre-compile slag worden de SQL-statements omgezet in Java-code, die vervolgens gecompileerd kan worden tot Java-bytecode. Deze pre-compile slag is volledig transparant gerealiseerd in JDeveloper. Daarnaast biedt JDeveloper een wizard waarmee bestaande PL/SQL procedures ontwikkeld in Developer gewrapped kunnen worden in Java. Hiermee is het mogelijk om enige mate van hergebruik te bewerkstelligen.



Figuur 7 JWeb-cartridge

Middels JDeveloper is het mogelijk een tweetal Cartridges te ontwikkelen: JWeb- en JCorba-cartridges. JWeb cartridges (zie figuur 7) betreffen Java Servlets, Java Applets die aan de Application Server-kant draaien en op basis van een HTTP-request vanuit de webbrowser de benodigde processing uitvoeren. Hierbij valt te denken aan interpreteren van het HTTP-request, het uitvoeren van PL/SQL, het raadplegen van de database, het opmaken van een HTML-pagina en het vervolgens terug sturen van de HTML-pagina naar de webbrowser. In feite betreft dit een 2de+ generatie internetoplossing [2]. Eén van de belangrijkste voordelen van deze aanpak betreft het feit dat het standaard HTTP-protocol gebruikt wordt. Met behulp van dit protocol is het eenvoudig mogelijk om door de betreffende firewalls die voor de webbrowser staan te komen. Een nadeel betreft onder andere het feit dat middels HTML geen geavanceerde gebruikersinterfaces te ontwikkelen zijn. Het *stateless* probleem doet zich voor, dat wil zeggen dat de toestand van de interactie tussen de webbrowser en de webserver niet bewaard wordt. Hierdoor is het niet mogelijk om meer geavanceerde internettoepassingen te ontwikkelen.



Figuur 8 JCorba-cartridge

Echte derde generatie internettoepassingen kunnen gebouwd worden middels JCorba-cartridges. JCorba-cartridges betreffen Java Applets (of Java Servlets) die zowel aan de webbrowser- als aan de Application Server-kant draaien (zie figuur 8). Deze Java Applets communiceren onderling middels IIOP. De communicatie met de Universal Data Server vindt plaats middels Oracle's SQL*Net protocol.

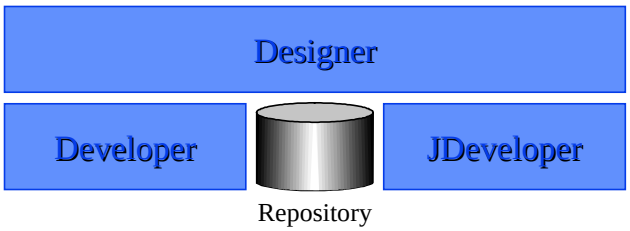
Universal Application Server

De Universal Application Server is een belangrijk onderdeel van de infrastructuur waarop de gebouwde cartridges draaien. De huidige versie van de Universal Application Server is, is versie 4.0 en bevat de implementatie van ICX voor de communicatie tussen cartridges, JavaSoft's JVM voor de interpretatie van in de JDeveloper gerealiseerde cartridges en een PL/SQL interpreter voor de interpretatie van cartridges geschreven in Oracle proprietary PL/SQL-taal. Voor de realisatie van ICX middels IIOP heeft Oracle gebruik gemaakt van de ORB (VisiBroker) van Inprise.

Universal Data Server

De Universal Data Server bevat naast de elementen van de Universal Application Server ook een SQL interpreter voor de ontsluiting van gegevens uit de database middels SQL. In feite is de Universal Data Server Oracle's traditionele rdbms met een aantal extra voorzieningen. Momenteel is de Universal Data Server nog niet beschikbaar, derhalve is het dus nog niet mogelijk om Java cartridges in de database te draaien, omdat ICX en de JVM nog niet geïncorporeerd zijn.

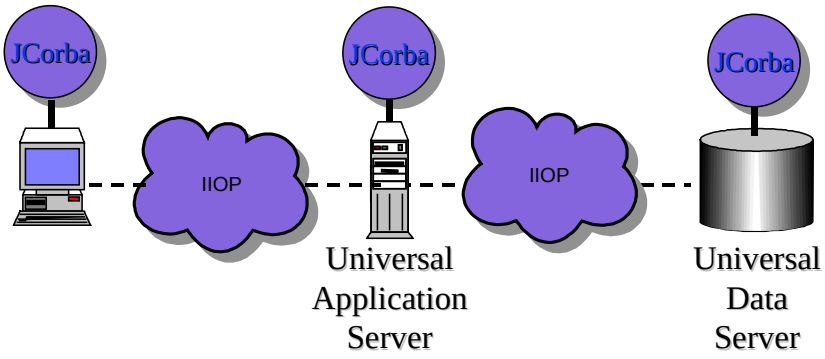
Toekomst
In de toekomst wil Oracle de aansluiting van Designer op JDeveloper realiseren. Op deze wijze is Oracle in staat om een het productaanbod te stroomlijnen (zie figuur 10) door één tool voor analyse en design te leveren: Designer. Vanuit Designer is het vervolgens mogelijk om de in kaart gebrachte applicatie in of Developer of JDeveloper te ontwikkelen.



Figuur 10 Productstrategie

Daarnaast is Oracle van plan om Designer, Developer en JDeveloper op één centrale repository aan te laten sluiten: de Oracle repository.

Als alles volgens planning verloopt komt eind 1998 de Universal Data Server uit: Oracle 8i. Deze bevat naast de huidige database functionaliteit: een PL/SQL- en SQL-interpreter, ook een JVM, een implementatie van de ICX en file server functionaliteit.



Figuur 9 300% Java

Middels de implementatie van een JVM en ICX in de database maakt het mogelijk om JCorba cartridges in alle lagen te gebruiken die middels IIOP communiceren (zie figuur 9). Oracle noemt dit het 300% Java initiatief.

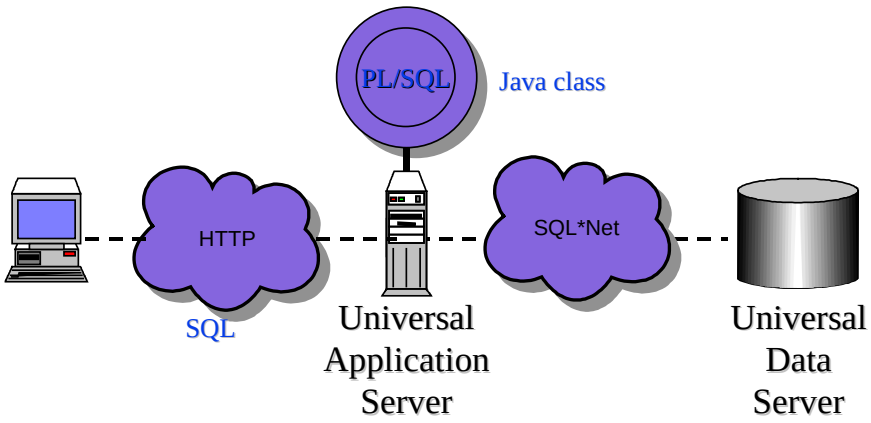
De incorporatie van file server functionaliteit in de Oracle database blijft verrassend. Het enige argument dat geldig is, is het feit dat Oracle hiermee de aanval op Microsoft's Windows NT kan openen. Echter, is het aan te raden om een database-server te gebruiken voor het opslaan van gegevens en een applicatie-server te gebruiken voor het uitvoeren van applicatie-logica. Beide producten dienen onafhankelijk op een solide infrastructuur, zoals bijvoorbeeld Windows NT of Unix, te draaien. In plaats van steken van energie in het bestrijden van Microsoft, zou Oracle liever op eerder genoemde punten zijn energie investeren om zo de achterstand in te halen.

Een andere ontwikkeling die Oracle heeft ingezet is de incorporatie van het Enterprise Java Beans (EJB) concept binnen de Universal Application Server en de Universal Database. Enterprise Java Beans betreffen Application Server en ontwikkelomgeving onafhankelijke componenten. Deze componenten

draaien in een zogenaamde EJB-container. Door de implementatie van een EJB-container in de Universal Application Server en de Universal Database van Oracle is het mogelijk om EJB te draaien binnen het NCA-concept.

Eén van de zwakke schakels in het totale productaanbod betreft de aansluiting van andere ontwikkelomgevingen, zoals bijvoorbeeld een Cobol of C++ ontwikkelomgeving, op het NCA-concept. Dit noopt het gebruik van JBuilder en derhalve Java in alle lagen de applicatie. Een nadeel van Java aan de server-kant betreft de performance die geleverd kan worden voor bijvoorbeeld zware batch-processing. Het gebruik van zogenaamde JIT[1]-compilers blijkt enige verbetering te bewerkstelligen doch hiermee kan niet altijd voldoende performance gehaald worden. Een nieuwe stroming die momenteel door IBM wordt ingezet, betreft het compileren van Java-bytecode naar native code middels de High Performance Compiler[2]. Hiermee wordt vergelijkbare performance met bijvoorbeeld native gecompileerde C of Cobol verkregen. Daarnaast zou door gebruik te maken van ontwikkelomgevingen voor bijvoorbeeld Cobol of C++ voor de bouw van cartridges aan de server-kant de benodigde performance gerealiseerd kunnen worden [3].

Migratie
Om een zo soepel mogelijke overgang van het gesloten, database-based en 2-tier client/server te typeren Designer/Developer strategie naar de meer open, objectbus-based, multi-tier client/server strategie van het NCA-concept, levert Oracle de mogelijkheid om vanuit JDeveloper PL/SQL te wrappen tot een Java class middels een zogenaamde “PL/SQL2Java” Wizard.

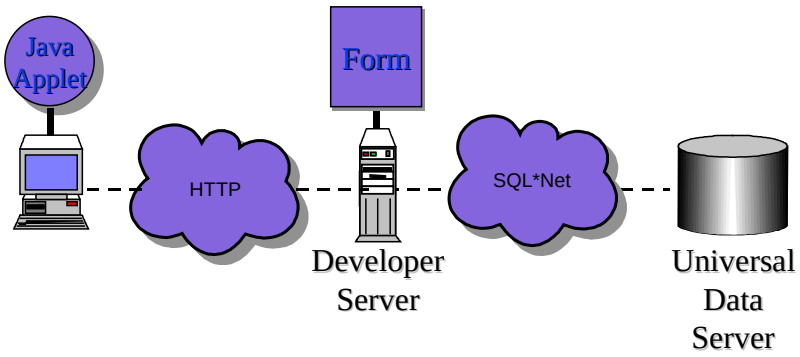


Figuur 11 Wrappen van PL/SQL in een Java class

Op deze manier is het mogelijk om bestaande logica te hergebruiken in NCA-applicaties, doordat aan de kant van de Universal Application Server de wrapped PL/SQL als cartridge draait. Deze is vervolgens middels ICX te benaderen vanuit andere cartridges. Of deze mogelijkheid in praktijk de benodigde performance kan leveren, valt te bezien. Feit, blijft dat met een geïnterpreteerde Java class die zelf PL/SQL interpreteert een enorme overhead gecreëerd wordt

Daarnaast biedt Oracle de mogelijkheid om vanuit Developer applicaties via het Internet te ontsluiten middels een Java applet aan de client-kant en de Developer Server aan de Application Server kant. De Java Applet aan de client-kant ontvangt zogenaamde “stuurcodes” van de Developer Server en bouwt op basis hiervan de presentatie op. Aan de kant van de Developer Server draait de originele presentatie middels Forms. De Developer Server zorgt ervoor dat alle presentatie zaken middels stuurcodes doorgegeven wordt aan de Java Applet. Vanuit de originele Form gezien, verandert er niets (zie figuur 10).

[1] Just In Time compiler: Nadat de Java Bytecode middels het HTTP-protocol naar het betreffende platform getransporteerd is, vindt er geen interpretatie meer plaats door de JVM maar wordt de Java Bytecode naar native uitvoerbare code gecompileerd. De JIT-compiler zorgt hiervoor en neemt derhalve de feitelijke taak van de JVM over [3].
[2] De High Performance Compiler wordt geleverd in Visual Age for Java 2.0.



Figuur 12 Developer Server

In de praktijk blijkt deze optie in bijvoorbeeld een WAN-omgeving met simpele P.C.'s (< Pentium) niet realiseerbaar te zijn met een goede performance.

Op basis van boven geschetste mogelijkheden voor de migratie naar NCA lijkt het er op dat de toekomstige rol van Designer zeer marginaal zal zijn.

Auteur

Drs. P.J. Koning is werkzaam bij Cap Gemini Technology Consulting.

Literatuur

[1] — "Multi-tier"-architectuur biedt flexibiliteit - Onderhoudbare informatievoorziening met behulp van softwarecomponenten, *P.J. Koning, G.M.A. van der Harst en G.F. Florijn*, Computable, 7 februari 1997.

[2] — Eureka voor e-handel - Objectbus en objecten maken Internet zakelijk Walhalla, *Drs. P.J. Koning*, Computable, 30 januari 1998.

[3] — De vuist van EBM in de strijd om het internet, Drs. P.J. Koning, Software Release Magazine, mei 1998.