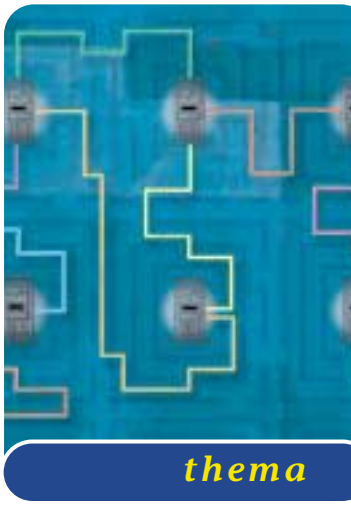
Zoals met meer dingen in onze branche gebeurt, vormt het geheel rondom "het opslaan en opvragen van (Java) objecten" onderwerp van menig verhitte discussie. "Moeten we gebruik maken van EJB/Entity Beans of toch maar gewoon van Plain Old Java Objects (POJOs)? Gebruiken we EJB/Entity Beans op basis van Container Managed Persistence (CMP) of Bean Managed Persistence (BMP)? Gaan we gebruik maken van Java Data Objects (JDO)?". Los van deze "low-level" keuzes lijkt men het over één ding wel eens te zijn; het nut van het gebruik van een Persistency Framework. In dit artikel licht Jan Vissers twee van deze Persistency Frameworks toe die door Oracle worden geleverd: Oracle9*i*AS TopLink en Business Components for Java.



TopLink en BC4J

Ontwikkelen vanuit datamodel of classmodel

In een Java applicatie worden instanties van classes gemaakt, om vervolgens iets zinnigs met deze objecten te doen. In de meeste gevallen is het wenselijk dat bepaalde gegevens, die door de objecten worden gerepresenteerd, op te slaan voor later (her)gebruik. Het bewaren en opnieuw construeren van een verzameling objecten (*object graph*) maakt onderdeel uit van de persistency functionaliteit van een Java applicatie. Momenteel wordt voor het overgrote deel van Java/J2EE applicatieontwikkeling voor wat betreft deze *persistency* een relationele database gebruikt als primaire *datasource*. Om deze relationele databases in te kunnen zetten, is het dus wel een eerste vereiste dat een Java applicatie een database kan benaderen en gebruiken.

Java Database Connectivity - JDBC JDBC is een Application Programming Interface (API) met als doel om in Java geschreven programma's te kunnen laten werken met relationele databases. De API bestaat uit een verzameling van Java classes en interfaces, waarvoor de standaard gedefinieerd is door Sun Microsystems. Een individuele provider of leverancier van een database, kan deze standaard uitbreiden met zijn eigen implementatie van de JDBC driver. Van deze driver implementaties wordt minimaal verlangd dat zij de ANSI SQL-92 Entry Level specificatie ondersteunen. Op deze manier ontstaat de mogelijkheid om SQL statements tegen een bepaalde database uit te voeren.

JDBC is echter een relatief *low-level* manier om relationele database te gebruiken in Java applicaties. JDBC biedt slechts een dun toegangslaagje naar een database. Dit manifesteert zich vooral door de hoeveelheid code, die nodig is om "iets" gedaan te krijgen. Voorts wordt het directe gebruik van JDBC voor *persistency* vaak beschouwd als minder geschikt binnen een verder sterk objectgeoriënteerde (OO) aanpak. Dit vormt de zogenaamde *impedance mismatch*. Al snel ontstaat de behoefte aan een hoger abstractieniveau.

Patterns JDBC abstractie begint meestal met de realisatie van een *classmodel*. De bedoeling hiervan is om een brug te slaan tussen de relationele en de objectgeoriënteerde wereld. Een verzameling van interfaces en classes is het gevolg, waarin zoveel mogelijk low-level zaken zijn afgeschermd. Dit is in de meeste gevallen echter niet het eindstation.

De mate van abstractie vult op dat moment nog maar een klein gedeelte in van de noodzakelijke, ondersteunende logica (*application plumbing*). Hoe minder de uiteindelijke applicatie zich hoeft aan te trekken van de opslag, des te beter. In diverse iteratieve slagen zal abstractielaag op abstractielaag gezet worden. Gedurende dit proces blijkt al snel dat bepaalde gelijksoortige problemen telkens weer opdoemen en opgelost moeten worden. De implementatie van deze *patterns* vormen samen met de JDBC abstractie uiteindelijk een *persistency framework*.

Persistency Framework Een persistency framework biedt een hoger abstractieniveau, waardoor de

details van de toegang tot objecten in een datasource verborgen zijn. Met behulp van een dergelijk framework kan een *persistency layer* geïmplementeerd worden. Deze laag beschrijft de domeinobjecten, het (data)*Model* van de uiteindelijke applicatie. Aan de onderkant van deze laag wordt via JDBC de database ontsloten en worden resultaten opgeleverd in een relationeel formaat. Aan de bovenkant ervan wordt door het persistency framework de "vertaling" gemaakt naar de domeinspecifieke objecten van de applicatie. Bij J2EE applicaties wordt deze persistency laag gebruikt als onderdeel van een Model-View-Controller (MVC) architectuur. Zoals al in een eerder artikel in "Java Magazine" werd beschreven[1], bestaan er tal van open source frameworks die ingezet kunnen worden om de persistency vraag te beantwoorden. In dit artikel kijken we echter naar twee frameworks die door Oracle worden geleverd; Oracle9iAS TopLink en Business Components for Java. Business Components for Java (BC4J) maakt al geruime tijd onderdeel uit van de Oracle toolstack. Oracle9*i*AS TopLink is daar nog niet zo heel lang geleden als nieuw alternatief aan toegevoegd.

> Hoe minder de uiteindelijke applicatie zich hoeft aan te trekken van de opslag, des te beter

Business Components for Java (BC4J)

BC4J wordt vooralsnog vooral gezien als een proprietary Oracle oplossing. Toegegeven, BC4J is al geruime tijd een onderdeel van de Oracle toolstack, maar is door zijn open, op Java/XML gebaseerde karakter zeer goed inzetbaar in een willekeurige J2EE setting. Daarnaast vormt BC4J meer dan alleen een persistency framework. Door onder andere een uitgebreide set aan geïmplementeerde J2EE design patterns en een geavanceerde JSP taglibrary wordt ondersteuning geboden voor applicatieontwikkeling op meerdere aspecten dan alleen persistency.

BC4J ondersteunt zowel het werken vanuit een bestaand datamodel als vanuit een classmodel. De eerste variant is de manier waaraan BC4J vooral zijn populariteit te danken heeft binnen de Oracle community. De tweede variant, het werken vanuit een classmodel, is een optie die vooralsnog veel minder vaak gebruikt wordt. Toch is dit, vooral ook door de groeiende ondersteuning van UML modelers in Oracle JDeveloper[2], het overwegen meer dan waard. Persistency op basis van een datamodel of op basis van een classmodel, is weer een van die onderwerpen, waar lang en geëmotioneerd over gediscussieerd kan worden. De OO/J2EE adept zal zeggen: "*het classmodel bepaalt het datamodel*". Iemand met een meer traditionele achtergrond zal vooral willen uitgaan van het RDBMS als bekende factor met faciliteiten voor snelle, geoptimaliseerde datatoegang. Het is vooral deze tegenstelling die voor de OO/J2EE groep aanleiding zal zijn om toch vooral eerst naar een framework als Oracle9iAS TopLink te kijken.

Oracle9*i*AS TopLink

Aan de basis van Oracle9*i*AS TopLink ligt het vroegere TopLink. Dit framework, ooit een product van ***T**he **O**bject **P**eople Ltd.* is in juni 2002 door Oracle overgenomen van WebGain. TopLink heeft een behoorlijk groot marktaandeel in de Java community. Een groot deel van deze community prefereert het werken vanuit een *classmodel* boven het werken vanuit een *datamodel*. Het is vooral deze *top-down* benadering die Oracle9*i*AS TopLink interessant maakt voor deze groep ontwikkelaars. In de nu volgende paragrafen zullen we nader ingaan op Oracle9*i*AS TopLink.

Oracle9*i*AS TopLink Onderdelen

De onderdelen van Oracle9iAS TopLink worden gevormd door een verzameling van Java-classes en -library's en de Oracle9iAS TopLink Workbench.

De classes en library's worden door de uiteindelijke Java applicatie op runtime gebruikt om op abstracte manier toegang te verkrijgen tot een datasource. Met behulp hiervan kan *persistency* worden toegevoegd aan Java applicaties. Hoe dit dient te gebeuren is voor een belangrijk deel beschreven in een persistency blauw-

Afbeelding 1. Dit classmodel wordt in de beschreven applicatie gebruikt.

1 "Je eigen O/R mapping onder controle - Inzet van open source frameworks" door Marc Portier (Outerthought); Java Magazine 1 - april 2003.

2 Dit is de Java/J2EE ontwikkelomgeving van Oracle.

druk van het classmodel. De *workbench* biedt, als onderdeel van de Oracle9*i*AS TopLink oplossing, de functionaliteit om een dergelijke persistency blauwdruk te maken.

ORACLE9IAS TOPLINK CONCEPTEN De Oracle9iAS TopLink concepten zijn redelijk eenvoudig en worden op hoofdlijnen gevormd door de volgende onderdelen:

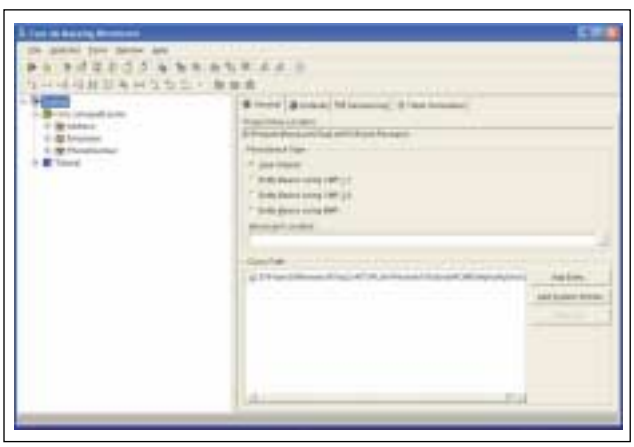
- Descriptors
- Mappings
- Database Sessions
- Units of Work

De *descriptors* en *mappings* maken deel uit van de al eerder genoemde blauwdruk van het classmodel. *Database Sessions* en *Units of Work* zijn voornamelijk Oracle9*i*AS TopLink runtime concepten. Laten we kort bij deze onderdelen stil staan.

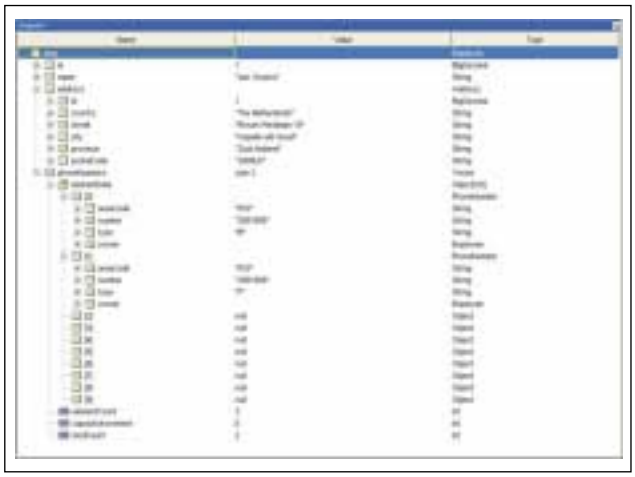
Descriptors

Een descriptor bevat informatie over hoe een object, inclusief eventuele relaties naar andere objecten, afgebeeld moet worden op een datasource. De volgende zaken maken deel uit van een descriptor:

- *Java class naam*
  Informatie over de class die beschreven wordt en de tabellen waarin instanties van deze class opgeslagen moeten worden.
- *Primaire sleutel*
  De te gebruiken primaire sleutel van de tabel met instanties.
- *Query sleutels*
  Een lijst van aliases voor de veldnamen.
- *Mappings*
  Waarin de attributen van de class beschreven worden, inclusief eventuele relaties van attributen met andere classes.



AFBEELDING 3. De Workbench toont welke domein classes onderwerp moeten zijn van het project



AFBEELDING 2. Een object graph

- *Property's*
  Met daarin aanvullende informatie over (het gebruik van) de descriptor. Denk hierbij aan zaken als; autorisatie, validatie.

Mappings

Een *mapping*, als onderdeel van een descriptor, wordt gebruikt om het toegangspad voor een objectattribuut aan te geven. Met andere woorden hoe wordt een objectattribuut afgebeeld in de uiteindelijke datasource. Hierbij wordt onderscheid gemaakt tussen twee typen mappings, namelijk:

- *Direct mappings*
  Het betreft hier een objectattribuut wat **1:1 ("direct-to-field")** of **1: 1' ("transformation type")** afgebeeld wordt in de datasource. De eerste groep kenmerkt zich doordat een Java type eenvoudig (direct) kan worden afgebeeld op het typesysteem van de datasource, bijvoorbeeld op een NUMBER of VARCHAR2 in geval van Oracle. De tweede groep vergt een vertaling van het Java type naar een ander, voor de datasource bruikbaar, type alvorens tot opslag kan worden overgegaan.
- *Relationship mappings*
  Dit zijn objectattributen die gerepresenteerd worden door referenties naar andere (TopLink enabled) classes op te nemen. Hierbij zijn twee varianten mogelijk, namelijk: **1:1** (een link van 1 class naar een andere class), of **1:n**, **m:n** (een verzameling van links).

Net zoals bij descriptors het geval is, geldt bij mappings dat ze aangemaakt en onderhouden kunnen worden met behulp van de workbench. Hiermee wordt de blauwdruk van het classmodel gemaakt ten behoeve van de persistency laag.

Database Sessions

Een *Database Session* beschrijft het verband tussen de applicatie en de database en vormt daarmee de link naar

de datasource. Dit object wordt ook gebruikt voor het lezen van gegevens uit de datasource.

Unit of Work

Een *Unit of Work* (letterlijk: "*eenheid van werk*") wordt in de Oracle9*i*AS TopLink context gebruikt om operaties die gegevens manipuleren (insert/update/delete) te kunnen groeperen.

WERKEN MET TOPLINK Nu de basisconcepten van Oracle9*i*AS TopLink duidelijk zijn, kunnen we aan de hand van een klein voorbeeld het werken met Oracle9*i*AS TopLink verder beschrijven. Hierbij zal aan een programma waarin gewerkt wordt met een bestaand *classmodel* (zie afbeelding 1), functionaliteit toegevoegd worden om gegevens weg te schrijven en op te halen. Het classmodel beschrijft de diverse domeinobjecten die weggeschreven en opgehaald dienen te worden. Let op: deze applicatie is niet in staat om runtime gegevens op te slaan en biedt dus geen persistency.

```
package com.cumquatit.acme;

public class TestRun  {
  public static void main(String[] args) {
     Employee emp = new Employee();

     Address empAddress = new Address();
     empAddress.setCity("Capelle a/d IJssel");
     empAddress.setCountry("The Netherlands");
     empAddress.setId(new java.math.BigDeci-
     mal("1"));
     empAddress.setPostalCode("2909LD");
     empAddress.setProvince("Zuid-Holland");
     empAddress.setStreet("Rivium Westlaan
     19");

     PhoneNumber businessPhone = new Phone-
     Number("B","010","2881690");
     PhoneNumber businessFax = new Phone-
     Number("F","010","2881688");

     emp.setAddress(empAddress);
     emp.addPhoneNumber(businessPhone);
     emp.addPhoneNumber(businessFax);
     emp.setId(new java.math.BigDecimal("1"));
     emp.setName("Jan Vissers");

     System.out.println(emp);
  }
}
```

Indien we het bovenstaande programma uitvoeren, zal op een gegeven moment een situatie ontstaan waarbij een *object graph* aanwezig is. Afbeelding 2 toont een momentopname van het programma. Als we hier niets aan doen, dan gaan de hierboven getoonde gegevens verloren op het moment dat het programma afgelopen is. Er zijn immers geen voorzieningen aanwezig om de gegevens vast te leggen.

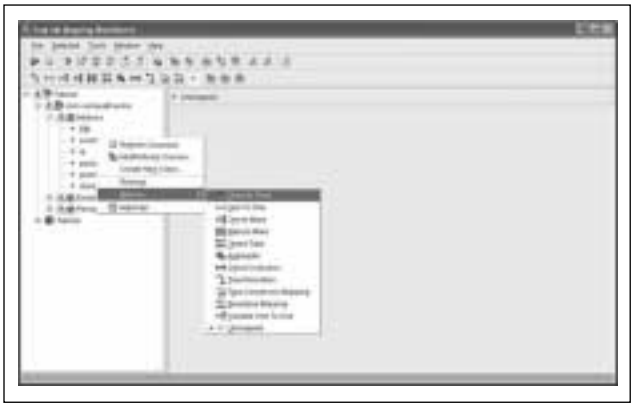
AFBEELDING 4. Met behulp van de Workbench kunnen de benodigde tabellen worden aangemaakt op basis van de te persisteren classes.

OPSLAG EN TOEGANG Om deze functionaliteit wel te bieden kan het *classmodel* gebruikt worden als basis voor een *persistency layer*. Met behulp van de Oracle9*i*AS TopLink Workbench kunnen we een project aanmaken met daarin alle, voor opslag en toegang, belangrijke aspecten. Deze informatie kan vervolgens op runtime gebruikt worden om de gewenste persistency functionaliteit in te vullen (zie afbeelding 3)

Om te beginnen kunnen we naast dat we informatie geven over de doelomgeving van de datasource, specificeren welke classes van het *classmodel* persistent gemaakt moeten worden. De hierboven getoonde afbeelding geeft aan dat de domein classes *Address, Employee* en *PhoneNumber* onderwerp moeten zijn van het project. Vervolgens kan nadere persistency informatie aan deze *descriptors* worden toegevoegd.

Eén van de zaken die hierbij aan de orde komt is hoe een *class* afgebeeld moet worden in de specifieke datasource. Dit is afhankelijk van het al dan niet aanwezig zijn van een bestaand datamodel in de vorm van (database) tabellen.

POINT AND CLICK Wanneer een datamodel al aanwezig is en gebruikt moet worden, dan kunnen de classes en bestaande tabellen via 'point-and-click' aan elkaar gekoppeld worden. Meestal zal het gebruik van TopLink echter direct verband houden met het feit dat een top-down benadering gekozen is voor persistency en zullen er geen bestaande tabellen aanwezig zijn. In dat geval kunnen met behulp van de Workbench de benodigde tabellen worden aangemaakt, op basis van de te persisteren *classes* (zie afbeelding 4). De volgende stap in het definitieproces is het vastleggen van afzon-



AFBEELDING 5. De afzonderlijke objectattributen worden beschreven in termen van fysieke opslag

derlijke *mappings* per descriptor. Zoals al eerder vermeld worden hierbij de afzonderlijke objectattributen beschreven in termen van fysieke opslag. Ook deze stap kan worden uitgevoerd met behulp van de workbench (zie afbeelding 5).

OMVANGRIJK ARSENAAL Voor de volledigheid wordt hier opgemerkt dat Oracle9iAS TopLink over een omvangrijk arsenaal van geavanceerde technieken beschikt om de persistency definitie van een *classmodel* vorm te geven. Een voorbeeld hiervan is de mogelijkheid om *relationship mappings* te definiëren op basis van *indirection*. Hiermee wordt voorkomen dat bij het teruglezen van een object, de complete *object graph* meteen wordt opgehaald.

Eventuele referenties (naar andere objecten) kunnen worden opgehaald, op een "need to know basis". Deze geavanceerde technieken, waaronder ook geoptimaliseerde caching en dergelijke behoren, vallen buiten het bereik van dit artikel.

BLAUWDRUK Op het moment dat het totale *classmodel* vastgelegd is in het project, kan de informatie klaargemaakt worden voor gebruik. Deze *deployment* stap resulteert in een XML of Java bestand. Dit eindresultaat bevat de complete blauwdruk van de *persistency laag* en wordt op runtime gebruikt. De volgende onderdelen verduidelijken het gebruik van deze blauwdruk in een Java applicatie.

Vastleggen van gegevens in een database

Ook hierbij wordt de eerder genoemde blauwdruk gebruikt. Merk op dat de hieronder afgebeelde broncode, dezelfde is als de oorspronkelijke broncode. Dit keer is echter de mogelijkheid tot opslaan van gegevens toegevoegd.

```
package com.cumquatit.acme;

import oracle.toplink.sessions.Database-
Session;
import oracle.toplink.sessions.Project;
import oracle.toplink.sessions.UnitOfWork;
import oracle.toplink.tools.work-
bench.XMLProjectReader;

public class TestRun  {
  public static void main(String[] args) {
     Employee emp = new Employee();

     Address empAddress = new Address();
     empAddress.setCity("Capelle a/d IJssel");
     empAddress.setCountry("The Netherlands");
     empAddress.setId(new java.math.BigDeci-
     mal("1"));
     empAddress.setPostalCode("2909LD");
     empAddress.setProvince("Zuid-Holland");
     empAddress.setStreet("Rivium Westlaan
     19");

     PhoneNumber businessPhone = new
     PhoneNumber("B","010","2881690");
     PhoneNumber businessFax   = new
     PhoneNumber("F","010","2881688");

     emp.setAddress(empAddress);
     );
     emp.addPhoneNumber(businessFax);
     emp.setId(new java.math.BigDecimal("1"));
     emp.setName("Jan Vissers");

     Project builderProject
     = XMLProjectReader.read("E:/Projects/Re-
     search/TopLink/JV001.xml");
     DatabaseSession session = builderPro-
     ject.createDatabaseSession();

     session.login("rnd_toplink","rnd_top-
     link");
     UnitOfWork unitOfWork =
     session.acquireUnitOfWork();
     Employee tempconsultant = (Employee)unit-
     OfWork.registerObject(emp);
  unitOfWork.commit();
  session.logout();
  }
}
```

De vetgedrukte regels tekst, is de toegevoegde code om opslag van gegevens mogelijk te maken. Details over hoe de diverse objecten inclusief attributen, opgeslagen moeten worden in een bepaalde database is onderdeel van de in de blauwdruk opgenomen definities.

Opvragen van gegevens uit een database

Hierbij zijn andermaal de vetgedrukte regels tekst, de

specifiek toegevoegde code om gegevens uit een database te halen. Hoe dit verder gebeurt, is verder voor de applicatie niet interessant en is vastgelegd in de blauwdruk.

```
package com.cumquatit.acme;

import oracle.toplink.sessions.Project;
import oracle.toplink.sessions.Database-
Session;
import oracle.toplink.expressions.Expression-
Builder;
import oracle.toplink.expressions.Expression;
import oracle.toplink.tools.workbench.XML-
ProjectReader;

import java.util.Vector;
import java.util.Iterator;

public class ReadDB  {
  public static void main(String[] args) {
     Project builderProject =
     XMLProjectReader.read("E:/Projects/Resear
     ch/TopLink/JV001.xml");
     DatabaseSession session =
     builderProject.createDatabaseSession();

     session.login("rnd_toplink","rnd_top-
     link");
     ExpressionBuilder builder = new
     ExpressionBuilder();
     Expression expression =
     builder.get("name").equal("Vissers");
     Employee employee =
     (Employee)session.readObject(Employee.cla
     ss,expression);

     System.out.println(employee);
     System.out.println(employee.get-
     Address());
     Vector phoneNumbers = employee.getPhone-
     Numbers();
     Iterator phoneIter  = phoneNumbers.ite-
     rator();
     while( phoneIter.hasNext() ) {
     System.out.println((Phone-
     Number)phoneIter.next());
     }

     session.logout();
  }
}
```

Heeft als resultaat:

```
Employee: Vissers
Address: Rivium Westlaan 19, Capelle a/d
IJssel, ZH, The Netherlands
PhoneNumber [BP]: (010) 288-1693
PhoneNumber [HP]: (010) 288-1690
```

BC4J EN TOPLINK Met BC4J en TopLink biedt Oracle twee alternatieven voor het realiseren van de persistency laag ten behoeve van een Java/J2EE applicatie. Op hoofdlijnen zijn beide producten in staat om dezelfde functionaliteit in te vullen. Het is voornamelijk de werkwijze van de ontwikkelaar die bepalend zal zijn, of voor BC4J dan wel voor TopLink gekozen wordt. In principe zijn beide oplossingen in staat om op dezelfde manier gebruikt te worden, dat wil zeggen; vanuit een classmodel of vanuit een datamodel. De groep die voor een aanpak kiest vanuit een classmodel, kenmerkt zich meestal door een relatieve onbekendheid met de door een RDBMS geboden functionaliteit. De groep die de voorkeur heeft om vanuit een datamodel te werken, is juist meestal afkomstig uit de, meer traditionele RDBMS-hoek. In dat licht kunnen BC4J en TopLink beschouwd worden als complementaire technieken. Toch is het gerechtvaardigd om BC4J iets hoger in de applicatieontwikkeling keten te plaatsen. Mede door de ondersteuning van vele andere J2EE design patterns, biedt het een extra niveau van abstractie. Vooral bij de ontwikkeling van MVC gebaseerde applicaties kan dit een welkome aanvulling zijn. Deze 'rangschikking' is samen met de manier waarop TopLink meer 'neutraal' toegang tot de datasource biedt, mogelijk aanleiding om BC4J gebruik te laten maken van TopLink. TopLink vormt hierin dan de relatief low-level O/R mapping (via POJOs, Entity Beans (CMP/BMP) of JDO) en BC4J[3] richt zich met name op de daadwerkelijke domeinobjecten, gebaseerd op de TopLink representatie.

3 BC4J ondersteunt O/R mapping op basis van POJOs en Entity Beans (CMP/BMP), maar nog niet op basis van JDO.

Jan Vissers is senior consultant bij Cumquat Information Technology. Cumquat richt zich op het bieden van oplossingen op het gebied van Self-Service applicaties, Portals, Webservices en Business-to-Business integratie, hierbij gebruikmakend van XML en Java technologie op basis van het Oracle platform.