Ewoud van den Boom, Senior Scientist, Operations Research Division NATO C3 Agency

Objectdatabases

OBJECTPERSISTENTIE ALTERNATIEVEN ONDERZOCHT

Wat is de populairste plaats om objecten op te slaan? In een relationele database! Dat is een merkwaardig gegeven. Men moet beseffen dat beide modellen, het relationele en het objectgeoriënteerde, elkaar niet zo goed verdragen. Deze fundamentele mismatch houdt de halve software-industrie daarom al decennia bezig. Vervolgens is er een groepje softwarefabrikanten dat objectdatabases maakt en beweert deze problemen op te kunnen lossen. Dit artikel is een korte verkenning in de objectrelationele problematiek en de wereld van objectdatabases aan de hand van een eenvoudige voorbeeldtoepassing met behulp van een concrete objectdatabase.

Iedereen die wel eens applicaties heeft geschreven en daarbij gebruik heeft gemaakt van een relationele database kent de ervaring: er komt verassend veel code aan te pas. Tegenwoordig is het eigenlijk zo dat er bijzonder veel tooling en API's aan te pas komen die de programmeur dit werk uit handen proberen te nemen. Van datalaag tot user interface, er is altijd wel iets dat de ontwikkelaar ondersteunt in het werken met relationele gegevens. Dit resulteert soms in gegenereerde code, een andere keer worden er op runtime allerlei zaken dynamisch geregeld, of allebei. In ieder geval zorgt het voor de nodige overhead en complexiteit. En ook al nemen deze hulpmiddelen de meeste ellende weg, het simpele feit dat ze nodig zijn onderstreept nogmaals de fundamentele incompatibiliteit tussen beide modellen. Bovendien valt geen van de oplossingen bij benadering als elegant te betitelen. De eerder genoemde incompatibiliteit wordt vaak de Object-Relational Impedance Mismatch genoemd. De reden waarom deze mismatch er is, heeft te maken met het feit dat het relationele denken uit de wereld van de wiskunde voortkomt en het object georiënteerde uit de wereld van de software engineering. Laten we de grootste pijnpunten eens nader bestuderen.

Een eerste probleem is dat de objectgeoriënteerde wereld het concept encapsulatie kent, en de relationele wereld niet. In OO zijn er beschermde, private gegevens die door de buitenwereld alleen door procedures op het object te beïnvloeden zijn. De functie van deze procedures is over het algemeen het waarborgen van de validiteit van het object. In de relationele wereld worden deze private gegevens publiek en zijn ze dus niet meer beschermd door de procedures, waardoor je data kwetsbaar worden. De objectgeoriënteerde wereld kent het concept van inheritance: gedeelde eigenschappen en gedrag worden geërfd van een gemeenschappelijke ouder. Dit voorkomt dubbele code en zorgt ervoor dat objecten, voor wat betreft gedeelde eigenschappen en gedrag, op dezelfde manier kunnen worden gebruikt. Dit laatste noemen we dan polymorfisme; dat trouwens ook op andere manieren dan inheritance kan worden bereikt. Inheritance valt niet één op één te vertalen naar een relationeel model.

Als een bepaald probleemdomein in een model wordt gevat, dan zal dit model, wanneer het is ontworpen door iemand die het relationele als uitgangspunt neemt, er anders uitzien dan wanneer het is ontworpen door iemand die objectoriëntatie als uitgangspunt neemt. Hier is een aantal oorzaken voor: wanneer men bijvoorbeeld classes modelleert, neemt men in het ontwerp ook gedrag mee. Daarnaast verschillen gegevensnormalisatieprincipes tussen data- en objectmodelleerdisciplines en bovendien verschillen in beide werelden de concepten van wat een relatie is. De beperkte ruimte die dit artikel biedt, staat niet toe alle verschillen, of de consequenties van deze verschillen, te benoemen. Het zal de lezer echter slechts weinig creativiteit vergen deze zelf te bedenken.

Oplossingen

Naast alle eerder genoemde, bestaande, niet elegante oplossingen die helpen de objectgeoriënteerde en relationele wereld aan elkaar te lijmen, en de dappere pogingen van de grote database-vendors om de objectgeoriënteerde wereld beter te faciliteren in de laatste generatie van hun producten, zijn er ook partijen die fundamentelere oplossingen nastreven. Zo zijn er mensen, bijvoorbeeld Eric Meijer[1] architect bij Microsoft, die de oplossing zien in een uitbreiding van bestaande programmeertalen. Er is in programmeertalenland echter veel discussie of dit een wenselijke richting is. In ieder geval: als dit er al komt moeten we nog een aantal jaren geduld hebben. WinFS is een andere poging om veel van de problemen weg te nemen. Hiervan kunnen we ook concluderen dat dit nog wel even op zich laat wachten. Daarnaast zijn er ook nog fundamentalisten, zowel in het object- als het relationele kamp, die vinden dat de andere stroming moet ophouden te bestaan.

Een directere, realistische, en bovendien bestaande oplossing, vinden we in het fenomeen objectdatabases. Het merkwaardige van deze technologie is dat iedere developer er wel eens van heeft gehoord, maar vrijwel geen enkeling heeft er ooit echt naar gekeken. Een korte introductie is daarom op zijn plaats. De eerste concepten van moderne relationele databases worden gevormd omstreeks 1970. Pas aan het einde van de jaren zeventig komen de eerste commerciële producten op de markt en pas tien jaar later begint de technologie flink aan acceptatie in het bedrijfsleven te winnen. De eerste concepten van moderne objectdatabases komen pas tot stand rondom 1980. Als de eerste commerciële objectdatabases op de markt verschijnen is de relationele database al tien jaar te koop en wordt deze net breed geaccepteerd. Dit zorgt er voor dat er in de markt geen enkele aandacht is voor de nieuwe technologie en de bedrijfjes die, de vaak dure, objectdatabases maken, komen niet tot ontwikkeling. Dit staat in schril contrast tot de relationele databases, die een spectaculair

commercieel succes genieten gedurende de jaren negentig en een industrie op zich zelf vormen. Anno 2005 kunnen we concluderen dat de relationele database de wereld bezit en de objectdatabase wegkwijnt in haar schaduw.

Veel mensen zijn zich niet bewust van deze geschiedenis. Vaak wordt ten onrechte verondersteld dat objectdatabases onsuccesvol zijn vanwege technische gronden. Als je iemand vraagt naar de reden waarom objectdatabases niet succesvol zijn is een vaak gehoord antwoord: performance. Dit is echter een absolute misvatting. Er zijn geen theoretische gronden waarom een objectdatabase per se slechter zou moeten performen dan een relationele database. Daarnaast wordt een en ander uiteindelijk volledig bepaald door de specifieke inplementaties van de databases en applicaties in kwestie. De makers van objectdatabases claimen dat ze vergelijkbaar of zelfs beter presteren dan hun relationele tegenhangers. Om de belofte van objectdatabases, namelijk transparante objectpersistentie en daarmee een dramatische reductie van complexiteit, te toetsen houden we een populaire, gratis objectdatabase voor het Microsoft .NET-platform tegen het licht.

db4o

Db4o is een open-source objectdatabase. Oorspronkelijk ontwikkeld voor de Java-wereld, heeft het zijn weg gevonden naar de .NET-wereld. Naast de Java- en .NET-edities zijn er ook versies beschikbaar voor de .NET compact runtime en Mono. De verschillende omgevingen kunnen zelfs elkaars objecten via de database gebruiken. Een van de eerste dingen die opvalt als je db4o downloadt en installeert, is dat het allemaal erg klein is. De download is slechts 3,3MB. De installatie-directory is ongeveer 6MB, maar de bulk van deze 6MB is source-code, documentatie en een (uitstekende) tutorial. De uiteindelijke 'database' is een enkele dll van 340KB. Er draaien geen serverprocessen. Er is geen 'enterprise manager'. Er zijn geen controlpanel-items te configureren. Er is geen schema-editor. Ook is er geen online 'bibliotheek'. Het mag duidelijk zijn dat werken met een objectdatabase een geheel andere ervaring zal zijn dan werken met de meeste relationele databases. Om een en ander te illustreren gaan we een eenvoudige applicatie ontwikkelen.

Werken met db4o

We maken een eenvoudige applicatie waarmee we de gegevens van een muziek CD-verzameling kunnen beheren. Het objectmodel bestaat uit slechts drie classes, en zijn geïmplementeerd als in codevoorbeeld 1.

Werkend met een objectdatabase is dit het belangrijkste onderdeel van het ontwerp en nu het er is kunnen we ons gaan richten op het werken met de objectdatabase. Zoals eerder gezegd, alles met db4o is eenvoudig. Dit geldt ook voor de API. Die is klein. De meeste ontwikkelaars kunnen al hun werk doen met behulp van slecht twee classes: ObjectContainer en ObjectSet; zie tabellen 1 en 2. Dit zijn ook nog eens twee objecten met een uiterst eenvoudige interface. Het volstaat om een referentie te zetten naar de db4o.dll.

De ObjectContainer is het centrale object dat toegang verschaft tot de database. De creatie ervan is eenvoudig, namelijk met behulp van een filename. Als het bestand in kwestie niet bestaat wordt een nieuw bestand aangemaakt. Het is ook mogelijk om de database remote te gebruiken met behulp van het ObjectServer-object (zie codevoorbeeld 2), maar daar ga ik in dit artikel verder niet op in.

Vervolgens creëren we twee artiesten die we toevoegen aan de database zoals in codevoorbeeld 3 is te zien. In tegenstelling tot de meeste relationele databases, hoeven we de objectdatabase niet van tevoren te vertellen hoe de informatie er uit zal zien die deze gaat bevatten. We kunnen instanties van de betreffende classes simpelweg aan de database aanbieden en die weet er verder wel raad mee.

```
public class Artist
{
  public Artist(){}
  public Artist(string name)
  {
    this.name = name;
  }
  private string name = null;
  public string Name
  {
    get{return this.name;}
    set{this.name = value;}
  }
  private ArrayList albums = new ArrayList();
  public ArrayList Albums
  {
    get{return this.albums;}
  }
}
public class Album
{
  public Album(){}
  public Album(Artist artist, string name)
  {
    this.artist = artist; this.Name = name;
    this.artist.Albums.Add(this);
  }
  private Artist artist = null;
  public Artist Artist
  {
    get{return this.artist;}
    set{this.artist = value;}
  }
  private string name = null;
  public string Name
  {
    get{return this.name;}
    set{this.name = value;}
  }
  private ArrayList songs = new ArrayList();
  public ArrayList Songs
  {
    get{return this.songs;}
  }
}
public class Song
{
  public Song(){}
  public Song(Artist artist, Album album, string name, int position)
  {
    this.artist = artist; this.album = album; this.Name = name;
    this.position = position; this.album.Songs.Add(this);
  }
  private Artist artist = null;
  public Artist Artist
  {
    get{return this.artist;}
    set{this.artist = value;}
  }
  private Album album = null;
  public Album Album
  {
    get{return this.album;}
    set{this.album = value;}
  }
  private string name = null;
  public string Name
  {
    get{return this.name;}
    set{this.name = value;}
  }
  private int position = 0;
  public int Position
  {
    get{return this.position;}
    set{this.position = value;}
  }
}
```

Codevoorbeeld 1. De implementatie van het objectmodel in db4o

ObjectContainer	
void activate(object obj, int depth)	Activeert alle members van het opgeslagen object tot de gespecificeerde diepte
bool close()	Sluit de ObjectContainer
void commit()	Commit de lopende transactie
void deactivate(object obj, int depth)	Deactiveert een opgeslagen object door alle members op null te zetten
void delete(object obj)	Verwijdert een opgeslagen object uit de database
ExtObjectContainer ext()	Geeft de ObjectContainer met een uitgebreidere interface terug
ObjectSet get(object template)	Query By Example (QBE) om objecten in de database te vinden
Query query()	Factory-method om Query-object te creëren
void rollback()	Maakt de lopende transactie ongedaan
void set(object obj)	Voegt nieuw object toe aan de database, een bestaand object wordt bijgewerkt met eventuele aangepaste gegevens.

Tabel 1. De class ObjectContainer

ObjectSet	
ExtObjectSet ext()	Geeft de ObjectSet met een uitgebreidere interface terug
bool hasNext()	Geeft true terug als er meer objecten in de set zitten
Object next()	Haalt het volgend object op uit de set
void reset()	Zet de cursor aan het begin van de ObjectSet
int size()	Geeft het aantal objecten in de set terug

Tabel 2. De class ObjectSet

```
// Create ObjectContainer object
ObjectContainer objectContainer = Db4o.openFile("c:\\test.odb");
```

Codevoorbeeld 2. Het ObjectServer-object

```
// Create Artists
Artist artistOne = new Artist();
artistOne.Name = "Mr. Bungle";
Artist artistTwo = new Artist();
artistTwo.Name = "Tomahawk";

// Add Artists to DB
objectContainer.set(artistOne);
objectContainer.set(artistTwo);
```

Codevoorbeeld 3. De creatie van twee artiesten

```
// Create prototype object
Artist prototype = new Artist();
ObjectSet objectSet = objectContainer.get(prototype);
// Write result to console
Console.WriteLine("All artists in db:");
while(objectSet.hasNext())
  Console.WriteLine(((Artist)objectSet.next()).Name);
// Let's retrieve all artists named Mr. Bungle
prototype.Name = "Mr. Bungle";
objectSet = objectContainer.get(prototype);
// Write result to console
Console.WriteLine("All artists name Mr. Bungle in db:");
while(objectSet.hasNext())
  Console.WriteLine(((Artist)objectSet.next()).Name);
```

Output

```
All artists in db:
Tomahawk
Mr. Bungle
All artists name Mr. Bungle in db:
Mr. Bungle
```

Codevoorbeeld 4. Het opvragen van alle artiesten uit de database

Het zou nu mooi zijn als we een query konden loslaten op de database om te kijken of de objecten goed zijn opgeslagen. Db4o biedt de ontwikkelaar twee query-mechanismen aan: Query By Example (QBE) en Simple Object Data Access (SODA). QBE werkt zeer eenvoudig. Je maakt een leeg prototypeobject en zet de eigenschappen van dit prototype zo neer dat deze overeenkomen met de eigenschappen van de objecten die we in de database willen vinden; als we alle instanties van een bepaald object willen vinden. De code in codevoorbeeld 4 toont hoe we met dit mechanisme alle artiesten uit de database opvragen en vervolgens hoe we een specifiek object vinden.
Dit geeft de volgende output, ook te zien in codevoorbeeld 4.

Met QBE kan men natuurlijk niet alle wenselijke queries maken. Dit verklaart het bestaan van SODA. SODA stoelt op dezelfde concepten als QBE, maar voegt daar een hoop zaken aan toe. Met SODA is het bijvoorbeeld ook mogelijk operatoren te gebruiken als: equals, not, smaller, greater, like, by identity, et cetera. Bovendien is het mogelijk callbacks te definiëren die worden aangeroepen als de query wordt uitgevoerd. Helaas staat de beschikbare ruimte in dit artikel ons niet toe in meer detail naar SODA te kijken.

Het is met een objectdatabase erg eenvoudig om objecten uit de database op te halen. Hierin schuilt echter een groot gevaar. In theorie is het mogelijk om met het ophalen van een enkel object de gehele database in het geheugen te laden, via directe en indirecte referenties die het object naar andere objecten heeft. Om deze, in potentie rampzalige, situatie te voorkomen kent db4o het concept van Activation Depth. In inactieve toestand bevat een object enkel de waarden van value type-eigenschappen. Zodra het object wordt geactiveerd, worden ook reference type-eigenschappen uit de database opgehaald. Deze functionaliteit is beschikbaar door de activate- en deactivate-methods op het ObjectContainer-object. Beide methods verwachten ook een depth-argument waarmee je kunt aangeven hoe diep het model moet worden geactiveerd. De default activation-depth die db4o hanteert is vijf. In theorie is een activation-depth van nul het snelst, maar dan moet je veel code schrijven die referenties activeert. Activation-depth is per database en per class te configureren en kan dynamisch worden aangeroepen voor objecten die al in het geheugen zijn. Laten we ons voorbeeld iets uitbreiden door een album en wat songs toe te voegen zoals in codevoorbeeld 5 is te zien.

Als we codevoorbeeld 5 uitvoeren gaat er iets mis. Het Album-object voegt zich in zijn constructor zelf toe aan de Albums-collectie van de Artist. Het Songs-object doet hetzelfde met de Songs-collectie van het Album-object. Vervolgens doen we een update van het Artist-object in de database. Als we nu de gegevens weer uit de database zouden laden, merken we dat er niets is toegevoegd aan het Artist-object. Dit wordt veroorzaakt door een mechanisme dat vergelijkbaar is met dat van Activation Depth. Om een update van een object niet te duur te maken zal de ObjectContainer het object maar één laag diep updaten; namelijk alleen de value type-eigenschappen van het object. Om db4o meer lagen (gerefereerde objecten) diep te laten updaten, moeten we dat vertellen zoals in codevoorbeeld 6 duidelijk wordt.

In tegenstelling tot Activation Depth is er van updateDepth geen dynamische variant en moet alle configuratie gebeuren voordat de

ObjectContainer wordt geopend. Update Depth kan per database en per class worden geconfigureerd. Het is ook mogelijk om met de cascadeOnUpdate-optie alle gerefereerde objecten bij te werken. We kunnen nu het opgeslagen object uit de database lezen zoals in codevoorbeeld 7 is te zien. Dit geeft de volgende output zoals ook te zien is in codevoorbeeld 7.

We hadden dezelfde output trouwens ook kunnen bereiken met codevoorbeeld 8.

Voor het verwijderen van objecten zijn dezelfde mechanismen aanwezig als voor het updaten, met dat verschil dat het niet mogelijk is een delete depth te specificeren. Er is dus alleen cascadeOnDelete dat de volledige objectenboom weggooit. Waar met updaten deze mechanismen nog bruikbaar zijn, schieten ze met deleten te kort. In ons voorbeeld willen we bij het verwijderen van een Album-object uit de database ook dat de bijbehorende songs verdwijnen, maar niet de artiest. Het is in db4o niet mogelijk om dit te configureren en het moet dus handmatig worden opgelost. Soms lukt dit door het model slim in elkaar te steken, maar dit werkt lang niet in alle situaties. De concepten van relationele integriteit bestaan niet in de wereld van OO. Db4o kan niet bedenken welke gerefereerde objecten mogen worden weggegooid en welke niet. Bovendien ziet db4o niet of er naar een object door geen enkel ander object meer wordt gerefereerd en daardoor wellicht betekenisloos is geworden. Db4o moet zich volledig baseren op de gegevens die reflection verstrekt over de classes die het moet opslaan en krijgt dus geen informatie over referentiele integriteit.

Het objectmodel in de voorbeeldapplicatie is erg eenvoudig. Db4o kan zonder problemen hele complexe objectstructuren in een keer opslaan, bewerken of veranderen. Maar het is belangrijk om te zien dat hier een valkuil ligt. De verleiding is groot om grote en complexe objecten te bedenken, die zelf weer referenties hebben naar andere complexe objecten en collecties met daarin wellicht nog meer complexe objecten. Deze grote en complexe objecten zijn op zichzelf geen probleem voor de database. Als we deze objecten zouden willen vertalen naar een relationeel model, zouden er heel wat tabellen aan te pas komen. Het ophalen, updaten of verwijderen van deze objecten in een relationele database zou lange en complexe SQL-statements met zich mee brengen die relatief duur zijn om uit te voeren. Het goede nieuws is dat de objectdatabase deze complexiteit wegneemt. Het slechte nieuws is dat er onder water nog steeds dure databaseopdrachten worden uitgevoerd. Objectdatabases maken het leven voor de programmeur aanzienlijk makkelijker, maar ze zijn niet in staat om op magische wijze dure databaseopdrachten goedkoop te maken.

Objectdatabases: bruikbare technologie

Db4o is een van de vele objectdatabases die beschikbaar is. Werken met db4o toont duidelijk de voordelen van het werken met een objectdatabase. De complexiteitsreductie in code valt zelfs spectaculair te noemen. Ook de hele omgeving van de database, tooling, serverprocessen, et cetera neemt dramatisch af in complexiteit. Belangrijke objectgeoriënteerde principes blijven bovendien intact: een Kat- en een Hond-object die erven van een Dier-object worden ook gevonden als je alle Dier-objecten uit de database leest. Property-procedures stellen de developers in staat private eigenschappen te beschermen. Over db4o kunnen we zeggen dat de database op een behoorlijk aantal platforms werkt, van Mac OS X tot Windows Mobile Edition en van Linux tot Windows. Bovendien is het gratis en is de sourcecode beschikbaar waardoor je ook kunt kijken hoe een en ander van binnen werkt.

Natuurlijk zijn er ook kanttekeningen te plaatsen. Voor wat betreft referentiële integriteit staat de ontwikkelaar er alleen voor; objectgeoriënteerde talen kunnen dit aspect niet uit zichzelf beschrijven. De maximale grote van de database is 256GB maar wordt al inefficiënt na 2GB (BLOBS worden in het bestandssysteem opgeslagen). Een ander punt, niet per se inherent aan de database, is dat het qua performance heel makkelijk en verleidelijk is om onverstandig met de database om te gaan. Het gevaar bestaat dat je met te veel, te grote, en te complexe objecten wilt werken. De ontwerper van een objectmodel moet blijven nadenken over hoe dit model zich gedraagt, en of het bruikbaar is in de context van een objectdatabase.

Vervolgens mogen we ook niet vergeten waarom objectdatabases nauwelijks worden gebruikt: marktacceptatie. Dit heeft direct als gevolg dat er nauwelijks 3rd party tools bestaan voor objectdatabases. Denk daarbij aan reporting tools, business analysis, datawarehousing-technologie, back-upsoftware, et cetera. Ook is expertise op dit gebied moeilijk te vinden, net als opleidingen en literatuur. Sommige mensen zullen je wijzen op het probleem dat de objectdatabase moet kunnen integreren met applicaties die een relationele database verwachten, of direct met een andere relationele database. Dan loop je tegen dezelfde impedance mismatch aan, alleen de andere kant op. Weer anderen zullen wijzen op het feit dat de data over het algemeen langer bestaan dan de applicatie en

```
// Create Album
Album newAlbum = new Album(artistOne, "Disco Volante");
objectContainer.set(newAlbum);

// Create Songs
Song song1 = new Song(artistOne,newAlbum,"Everyone I Went to
   High School With Is Dead",1);
Song song2 = new Song(artistOne,newAlbum,"Chemical Marriage",2);
Song song3 = new Song(artistOne,newAlbum,"Carry Stress in the Jaw",3);
Song song4 = new Song(artistOne,newAlbum,"etc",4);

// Update artist one
objectContainer.set(artistOne);
```

Codevoorbeeld 5. Het toevoegen van een album en enkele songs

```
// Instruct db4o to allways update Artist objects 3 layers deep
Db4o.configure().objectClass(typeof(Artist)).updateDepth(3);

// Create ObjectContainer object
ObjectContainer objectContainer = Db4o.openFile("c:\\test.odb");
```

Codevoorbeeld 6. Een diepe update van meer lagen door db4o

```
// Get songs from specified artist and print to screen
Artist mrBunglePrototype = new Artist("Mr. Bungle");
ObjectSet artists = objectContainer.get(mrBunglePrototype);
while(artists.hasNext())
{
   Artist actualMrBungleObject = (Artist)artists.next();
   foreach(Album album in actualMrBungleObject.Albums)
      foreach(Song song in album.Songs)
         Console.WriteLine(string.Format("Artist: {0}, Album: {1},
            Name: {2}, Position: {3}",
            actualMrBungleObject.Name, album.Name, song.Name,
            song.Position));
}
```

Output:

Artist: Mr. Bungle, Album: Disco Volante, Name: Everyone I Went to High School With Is Dead, Position: 1
Artist: Mr. Bungle, Album: Disco Volante, Name: Chemical Marriage, Position: 2
Artist: Mr. Bungle, Album: Disco Volante, Name: Carry Stress in the Jaw, Position: 3
Artist: Mr. Bungle, Album: Disco Volante, Name: etc, Position: 4

Codevoorbeeld 7. Het opgeslagen object uit de database lezen

```
// This gives the same result
Song mrBungleSongPrototype = new Song();
mrBungleSongPrototype.Artist = mrBunglePrototype;
ObjectSet mrBungleSongs = objectContainer.get(mrBungleSongPrototype);
while(mrBungleSongs.hasNext())
{
   Song song = (Song)mrBungleSongs.next();
   Console.WriteLine(string.Format("Artist: {0}, Album: {1}, Name: {2},
   Position: {3}", song.Artist.Name, song.Album.Name, song.Name,
   song.Position));
}
```

Codevoorbeeld 8. Dezelfde output als met codevoorbeeld 7

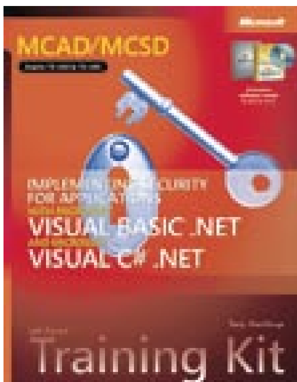
dat werken met objectdatabases de data te veel aan de applicaties binden. De uiteindelijke conclusie moet zijn dat objectdatabases een bruikbare technologie is die veel complexiteit verwijdert voor de ontwikkelaar. Of een objectdatabase een geschikte oplossing is voor een specifieke toepassing zal per situatie moeten worden beoordeeld.

Ewoud van den Boom is Senior Scientist bij de Operations Research Division van de NATO C3 Agency.

Nuttige internetadressen:
http://www.db4o.com/

Noten:
Meer informatie over een pogingen een brug te slaan tussen de wereld van objectgeoriënteerde (CLR), relationele (SQL) en hiërarchische data (XML) zie http://research.microsoft.com/~emeijer/
(Footnotes)

(advertentie Microsoft Press)



MCAD/MCSD Self-Paced Training Kit: Implementing Security for Applications with Microsoft® Visual Basic® .NET and Microsoft Visual C#® .NET
ISBN: 0-7356-2121-7
Auteur: Tony Northrup
Pagina's: 640