enterprise

In maart van dit jaar verraste Oracle vriend en vijand door de code van Oracle TopLink te doneren aan de Eclipse Foundation. Dit artikel zal een overzicht geven van het TopLink product, en vervolgens aan de hand van enkele voorbeelden laten zien hoe TopLink de EJB3 JPA-standaard implementeert en uitbreidt. Tenslotte zal aandacht besteedt worden aan het EclipseLink project en de rol die TopLink hierin zal spelen.

EclipseLink

TopLink werd Open Source

Jeroen van Wilgenburg en Peter Ebell
zijn werkzaam als consultants bij AMIS Services BV te Nieuwegein. E-mail: jeroen.van.wilgenburg@amis.nl en peter.ebell@amis.nl

De data die in applicaties gebruikt worden, komen veelal uit een database. Deze data moeten zo snel mogelijk naar objecten omgezet worden. Het is als Java-developer veel prettiger om met Java-objecten te werken en (bijna) niets van de database af te weten. Data die via JDBC uit een database opgevraagd worden, komen als ResultSet-object terug waar per rij doorheen gelopen moet worden. Vervolgens moet elke rij naar een object omgezet worden. Dit omzetten is een hoop werk. Het behoeft voor vrijwel elke applicatie en query dezelfde code. Voor dit herhalende klusje schieten ORM-frameworks te hulp. ORM staat voor Object-Relational Mapping. ORM-frameworks zijn er in talloze variaties, de simpelste zorgen alleen voor het mappen van objecten en meer geavanceerde frameworks doen aan caching en beheren de relaties tussen entiteiten die ook in de database te vinden zijn.

Historie

Eén van de oudste ORM-frameworks is TopLink. De eerste versies, die begin jaren negentig ontwikkeld werden door een bedrijfje genaamd "The Object People" (de "Top" in TopLink komt hier vandaan), waren geschreven in de OO-taal SmallTalk. Toen het midden jaren negentig duidelijk begon te worden dat het Sun menens was met hun nieuwe OO-taal "Java", werd besloten om TopLink in deze taal te herschrijven. In 1997 zag de eerste versie van "TopLink for Java" het licht, die al snel een commercieel succes werd. In 2000 werd "The Object People" overgenomen door WebGain, een bedrijf dat zich richtte op de ontwikkeltools markt en ervoor gezorgd heeft dat het product een steeds grotere populariteit kreeg. In 2002 was de marktpenetratie van TopLink zodanig, dat Oracle er een 'unique selling point' voor hun applicatieserver in zag, en TopLink (en daarmee vrijwel geheel WebGain) opkocht. Net als indertijd bij de overname van The Object People door Webgain werd uitdrukkelijk niet alleen de technologie aangeschaft, maar werden ook alle ontwikkelaars ingelijfd; de kern van het TopLink team bevat nog steeds veel mensen die er al vanaf het prille begin bij betrokken zijn. Onder de vlag van Oracle is het TopLink team verder gegroeid, en is het product nieuwe richtingen ingeslagen. Voorbeelden hiervan zijn de mogelijkheid om gegevens in XML te ontsluiten op dezelfde manier als gegevens in een relationele database (OXM mapping), en de actieve rol die het TopLink-development team heeft gespeeld bij het opstellen van de EJB3 JPA specificatie, hetgeen heeft geresulteerd in het feit dat TopLink Essentials nu Sun's referentie-implementatie van deze standaard is. Het meest recente (maar vast niet laatste) hoofdstuk in TopLink's geschiedenis is dat Oracle de source code heeft gedoneerd aan de Open Source community, waar het het uitgangspunt voor het ambitieuze 'EclipseLink'-project vormt.

Technologie

TopLink is niet alleen een van de oudste, maar ook een van de meest geavanceerde ORM-frameworks. Het is bedacht en jarenlang doorontwikkeld door mensen die TopLink veelvuldig 'in het veld' gebruikt en ondersteund hebben, en al deze kennis en ervaring met 'real life'-persistence problematiek is verwerkt in het product. Zonder te streven naar volledigheid volgt hier een aantal features van TopLink die dit illustreren:

** Caching*

Het belangrijkste wapen dat TopLink hanteert om de performance van J2EE-applicaties te maximali-

seren is de 'centrale cache' (genaamd 'Identity Maps'). Het idee hierachter is dat ieder (data) object zich slechts één keer in het geheugen bevindt, en door alle gebruikers (sessies) gedeeld wordt. Dit komt de performance op een aantal fronten ten goede: het bespaart geheugen, het voorkomt het steeds opnieuw initialiseren van objecten die dezelfde data bevatten (en de veelvuldige garbage collection die daar weer het gevolg van is), en tenslotte hoeven applicaties doorgaans minder vaak de database te raadplegen omdat er minder "stale data" is; mutaties op bestaande objecten zijn immers direct zichtbaar voor alle sessies.

** Transacties*

De enige uitzondering op de regel dat een object maar één keer in het geheugen voorkomt is als er een gebruiker(sessie) een transactie aan het uitvoeren is op dit object. Hij doet dit in een eigen 'transaction space', UnitOfWork genaamd, waarin hij door TopLink gecreëerde 'klonen' van de oorspronkelijke objecten kan manipuleren. Aan het eind van de transactie wordt de UnitOfWork gecommit, waarna TopLink automatisch de minimale set DML (insert, update en delete) statements bepaalt die nodig zijn om de wijzigingen die in deze transactie zijn gedaan naar de database te persisteren. Vervolgens worden de wijzigingen ook in de objecten in de 'centrale cache' doorgevoerd, zodat andere sessies deze direct 'zien'.

** Query Language*

TopLink heeft een zeer krachtige, object georiënteerde query language, die de ontwikkelaar in staat stelt om volledig in termen van zijn Object Model, zonder kennis van het onderliggende relationele data model in de database, zeer complexe queries samen te stellen. TopLink zorgt er vervolgens voor dat dit vertaald wordt naar efficiënte SQL statements.

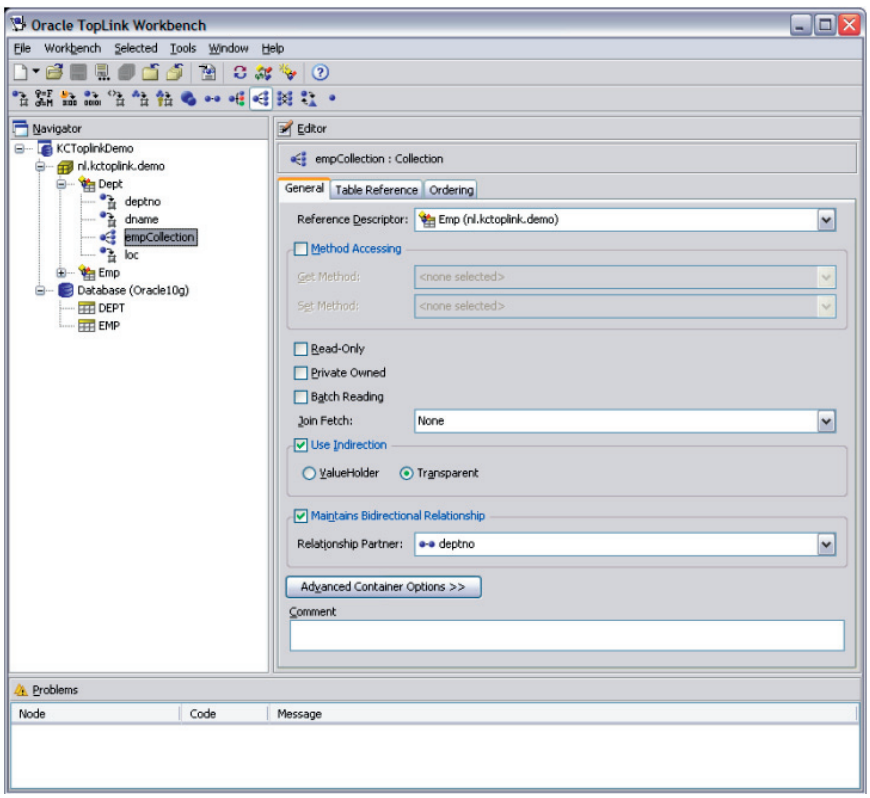
** Performance Tuning*

De performance van een persistence framework wordt in de praktijk voor een groot deel bepaald door hoe goed het runtime gedrag van het framework aan te passen is aan het specifieke, empirische gebruik van data door de applicatie. Welke data is voornamelijk statisch, waarop wordt veel gemuteerd, welke gegevens zijn altijd tegelijk nodig, welke data wordt door veel sessies tegelijk gebruikt, dit zijn slechts een aantal die van invloed zijn op hoe efficiënt een ORM framework zijn werk doet. En met name op dit gebied excelleert TopLink. Met talloze geavanceerde technieken, zoals indirection (soort van 'just-in-time' querying), partial attribute reads (alleen die velden van objecten invullen die je nodig hebt), batch reading (in één query de 'children' van een groot aantal 'parents' ophalen), en uitgebreide declaratieve en programmatische controle over cache afmetingen of strategieën biedt TopLink zeer uitgebreide mogelijkheden tot het fine-tunen van het runtime gedrag van het framework. Een zeer groot deel van deze technieken kunnen zodanig worden toegepast dat de impact op de applicatiecode minimaal is, hetgeen belangrijk is omdat er vaak een 'realistic load and usage' nodig is om zowel performance bottlenecks als het effect van tuning maatregelen waar duidelijk waar te nemen.

Figuur 1. TopLink historie.

** Designtime Tooling*

Alle ORM-frameworks maken gebruik van metadata, die de manier beschrijft waarop de vertaling



Figuur 2. TopLink Mapping Workbench

tussen het Object-model en het relationele database model moet plaatsvinden. Vaak wordt deze informatie vastgelegd in XML files en/of annotaties. TopLink heeft zich al in een vroeg stadium gerealiseerd dat deze informatie enerzijds cruciaal is voor de applicatie, en anderzijds dat het onderhouden van deze informatie lastig en zeer foutgevoelig werk kan zijn. Daarom is er een team dat zich exclusief bezighoudt met de 'Mapping Workbench', een krachtige, stand alone IDE die het zeer eenvoudig maakt om de meta-data aan te maken, te bekijken, te modificeren en (heel belangrijk) te valideren. Ook zijn er (gedeeltes van) de Mapping Workbench geïntegreerd met JDeveloper zodat je binnen één IDE het Object model, database ontwerp én de mapping ertussen kan onderhouden op een zo veel mogelijk declaratieve manier.

Persistence API

TopLink heeft altijd een eigen Java API gehad waarmee alle features en aspecten van het framework te bereiken en te gebruiken zijn. Door de jaren heen zijn er echter op standaarden gebaseerde persistence API's ontstaan zoals Java Data Object (JDO) en EJB container managed persistence (CMP), en TopLink heeft er altijd effort in gestoken om deze API's te ondersteunen of er zelfs speciale versies voor uit te brengen (TopLink for WebLogic en TopLink for WebSphere waren de CMP-versies van TopLink voor deze applicatieservers, later zijn alle versies weer samengebracht in één product). Desondanks zijn verreweg de meeste gebruikers altijd de 'native API' en POJO-based persistence blijven gebruiken. Reden hiervoor is dat de gestandaardiseerde API's nooit uitgebreid genoeg en te 'gesloten' waren om alle krachtige TopLink-features en mogelijkheden te ontsluiten. Ook het TopLink-team zelf heeft hun voorkeur voor POJO-persistence en het gebruik van hun eigen API nooit onder stoelen of banken gestoken, maar voelde zich als klein bedrijf daarbij vaak een roepende in de woestijn. De grote ommekeer hierin kwam toen de EJB 3.0 (JPA) specificatie geschreven zou gaan worden, en een aantal ontwikkelaars van het TopLink-team hier een trekkersrol bij zijn gaan vervullen.

JPA

De Java Persistence API (JPA) is een onderdeel van de Enterprise Java Beans 3.0 (EJB3) specificatie. In veel blogs en artikelen worden de termen EJB3 en JPA vaak door elkaar heengehaald. JPA is een subset van EJB3, in de meeste gevallen wordt met EJB3 alleen de JPA bedoeld.
Er zijn nu een aantal verschillende implementaties verkrijgbaar (TopLink, Hibernate, OpenJPA, Kodo, JPOX en een implementatie van SAP). De referentie implementatie die Sun aanbiedt is TopLink Essentials, de uitgeklede versie van de normale TopLink. Met de komst van EclipseLink zullen ook de uitbreidingen van TopLink vrij voor iedereen beschikbaar zijn.

Ter introductie van JPA volgt nu een voorbeeld van hoe eenvoudig data uit de database gehaald kunnen worden. Deze data worden uiteraard automatisch naar Java-objecten gemapt. Na de introductie zal een aantal TopLink-specifieke features gedemonstreerd worden.

persistence.xml

Het enige configuratiebestand is persistence.xml en is te vinden in de META-INF directory op het classpath. In deze file staan de details van de databaseverbinding of een verwijzing naar een datasource. Naast de databasedetails staan ook de gebruikte domeinobjecten in dit bestand. Voor dit voorbeeld is een Oracle database met het, voor Oracle-gebruikers, bekende SCOTT-schema. Voor de klasse Employee is de tabel EMP gebruikt.

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence  xmlns="http://java.sun.com/xml/ns/per-
sistence" version="1.0">
    <persistence-unit name="scott">
      <class>nl.amis.domain.Employee</class>
      <properties>
               <property  name="toplink.jdbc.driver"
value="oracle.jdbc.OracleDriver"/>
                 <property  name="toplink.jdbc.url"
value="jdbc:oracle:thin:@localhost:1521:orcl"/>
                <property  name="toplink.jdbc.user"
value="scott"/>
             <property name="toplink.jdbc.password"
value="tiger"/>
      </properties>
  </persistence-unit>
</persistence>
```

Annotaties

De mappings naar objecten kunnen in xml gemaakt worden, maar het is gebruikelijker om annotaties te gebruiken. De JPA-annotaties maken gebruik van configure by exception, dit betekent dat er pas annotaties (of attributen op de annotaties) gebruikt hoeven te worden als er afgeweken wordt van de standaard.

De klasse Employee.java ziet er als volgt uit:

```
@Entity
@Table(Name="EMP")
public class Employee {
  private int empno;
   private String ename;
   // rest van klasse bevat getters en setters
}
```

Met @Entity is aangegeven dat deze klasse een EJB is en dat er verder gekeken moet worden naar

TopLink heeft altijd een eigen Java API gehad waarmee alle features en aspecten van het framework te gebruiken zijn

annotaties op de klasse. Omdat de klassenaam Employee afwijkt van de tabelnaam moet de @Table annotatie gebruikt worden. Had de klasse EMP geheten dan was de @Table annotatie overbodig geweest. Omdat de namen van de kolommen empno en ename wel overeenkomen met de EMP-tabel hoeft de @Column annotatie niet gebruikt te worden. Het is echter wel toegestaan om annotaties te gebruiken. Dit kan de code duidelijker maken en vaak bevatten gegenereerde klasses ook alle annotaties.
Dit voorbeeld is simpel gehouden omdat er al veel blogs en artikelen over JPA geschreven zijn en de uitgebreide features van TopLink een stuk interessanter zijn. In een eerder in Java Magazine gepubliceerd artikel, geschreven door Lucas Jellema, wordt uitgebreider ingegaan op het gebruik van JPA (nr. 1/2006).

JPQL

Bij JPA hoort een querytaal genaamd JPQL die veel overeenkomsten met SQL vertoont, maar net even anders is. De taal lijkt misschien in eerste instantie wat overbodig, maar heeft als groot voordeel dat queries op java-objecten uitgevoerd worden en er geen kennis van de tabellen in de database nodig is. De EJB-implementatie vertaalt de queries in JPQL naar native-SQL.

```
EntityManagerFactory emf=Persistence.createEntityMan
agerFactory("scott");
EntityManager em = emf.createEntityManager();

Query query = em.createQuery("select e from EmpRs e
where e.empno>7800");
List<Employee> list = query.getResultList();
//doe iets met lijst
```

De EntityManagerFactory wordt aangemaakt op basis van de persistence-unit die in persistence.xml gedefinieërd is. Queries worden uitgevoerd op het queryobject wat verkregen is van een EntityManager.

NamedQuery

Een alternatief om een query te definiëren is een NamedQuery, hierbij wordt de query als een attribuut van de @NamedQuery annotatie gedefinieerd. Zet op de klassedefinitie van Employee de annotatie @NamedQuery(name = "emp2", query = "select e from EmpRs e where e.empno>7800") en instantieer het query-object als volgt:

```
Query query = em.createNamedQuery("emp2");
```

Op deze manier staan er niet kris-kras queries door de code, maar kunnen de queries netjes op een paar plekken verzameld worden. Om meerdere queries op een klasse te zetten moet de @NamedQueries annotatie gebruikt worden, dit is een array van @NamedQuery-annotaties.

```
@NamedQueries({
  @NamedQuery( name = "emp2", query=""select e from
EmpRs e where e.empno>7800"),
  @NamedQuery( name = "emp3", query=""select e from
EmpRs e where e.empno>7900"),
})
```

Tools

Doordat JPA een geaccepteerde standaard is loont het voor de IDE-developers om voor goede tools te zorgen. De vier grootste IDE's (Eclipse, IntelliJ IDEA, NetBeans en JDeveloper) bieden tegenwoordig goede ondersteuning voor EJB's en er hoeft nog maar weinig met de hand geconfigureerd te worden. Het mappen van tabellen naar klasses (inclusief relaties) gaat met een paar muisklikken. IntelliJ IDEA is zelfs zo intelligent om Strings die JPQL queries bevatten te valideren en te voorzien van autocompletion.

Verschillende implementaties

Zoals al eerder genoemd is er veel keus tussen verschillende implementaties. Een reden om voor een bepaalde JPA-implementatie te kiezen zou de extra mogelijkheden van die implmentatie kunnen zijn. Omdat de JPA-specificatie vrij beperkt is zul je al snel gebruik maken van implementatie-specifieke features. Een ander punt dat in de keus kan meewegen is de support, Oracle is bijvoorbeeld goed telefonisch te bereiken en Hibernate heeft een heel uitgebreid forum waar erg veel te vinden is.

Oracle TopLink is een krachtig ORM framework met een rijke historie

@Convertor en @Convert annotaties

De @Converter en @Convert zijn twee annotaties die geen onderdeel van de JPA-specificatie zijn en alleen in TopLink te vinden zijn. De @Converter annotatie wordt gebruikt als de data niet direct op een Java-object te mappen is (of andersom). Een veelgebruikte kolommapping is voor Booleans. Booleans worden in veel databases niet ondersteund en worden op talloze manieren in de database opgeslagen (0/1, J/N, Y/N, T/F etc.) en dit moet eigenlijk altijd op een java-boolean-type gemapt worden. Hier kan een Converter voor gebruikt worden. Een voorbeeld is een tabel met een veld visible (een char(1) in MySQL), dit veld kan de waarde Y of N bevatten en in Java moet dit gewoon als een boolean behandeld worden. Een goede kandidaat voor een Converter dus. De eerste stap is het veld visible van de juiste annotaties voorzien:

```
@Converter(name = "YnConverter", converterClass =
YnConverter.class)
@Convert(value = "YnConverter")
public Boolean getVisible() {
    return visible;
}
```

De @Converter annotatie is de definitie van de Converter, deze annotatie had ook op de klasse zelf kunnen staan, voor het geval de Converter op meerdere plekken gebruikt gaat worden. De name is een String en converterClass een klasse (er kan hier ook een fully qualified name gebruikt worden). De annotatie @Convert zorgt ervoor dat de daadwerkelijke conversie ook echt wordt uitgevoerd.

De volgende stap is de YnConverter aanmaken. Een Converter-klasse is een klasse die de interface oracle.toplink.mappings.converters.Converter implementeert. Voorlopig zijn de methodes convertDataValueToObjectValue en convertObjectValueToDataValue alleen van belang. De eerste methode zorgt ervoor dat databasevelden naar objectwaardes geconverteerd worden en de tweede methode doet het tegenovergestelde.

```
public  Object  convertDataValueToObjectValue(Object
object, Session session) {
    String yn = (String) object;
    if (yn.equalsIgnoreCase("Y")) {
        return true;
    } else {
        return false;
    }
}

public  Object  convertObjectValueToDataValue(Object
object, Session session) {
    Boolean yn = (Boolean) object;
    if (yn) {
        return "Y";
    } else {
        return "N";
    }
}
```

Het toevoegen van de @Convert annotatie zorgt ervoor dat overal in TopLink het veld visible als een boolean behandeld wordt. Er zal dan ook een foutmelding getoond worden als er in queries Y of N gebruikt wordt. Een voorbeeld van een geldige query is select object(b) from BoekEntity b where b.visible = true.

Andere annotaties

De @Converter en @Convert annotaties zijn een kleine greep uit de uitgebreide set uitbreidingen die TopLink te bieden heeft. TopLink biedt nog een aantal eigen datastructuren en uitgebreide locking en caching-mechanismes. Meer informatie over deze features (en een preview van TopLink 11g) is te vinden op http://www.oracle.com/technology/products/ias/toplink/preview/index.html

EclipseLink

De meest recente ontwikkeling met betrekking tot TopLink is er een die vriend en vijand heeft verrast: in maart van dit jaar heeft Oracle TopLink gedoneerd aan de Open Source community, meer concreet aan de Eclipse Foundation. Hiermee is Oracle 'board member' van de Eclipse Foundation geworden en heeft het zijn membership status verhoogd naar 'Strategic Developer'. Vervolgens heeft Oracle een Eclipse-project voorgesteld onder de naam Eclipse Persistence Platform, dat geleid zal worden door Oracle zelf. In dit project zal de gedoneerde code van TopLink een grote rol gaan spelen, maar het zal verder gaan dan enkel ORM en OXM mapping. Het doel van het Eclipse Persistence Platform is om één enkel framework te bieden met support voor een grote verscheidenheid aan persistence standaarden. Het zal de volgende componenten bevatten:

- EclipseLink-ORM biedt een krachtig Object-Relational Mapping framework met support voor JPA. Het zal niet als verrassing komen dat dit met de gedoneerde TopLink code wordt geïmplementeerd.
- EclipseLink-OXM biedt een krachtig Object-XML Mapping framework met support voor de Java API voor XML binding (JAXB). Ook deze component zal gebaseerd worden op TopLink.
- EclipseLink-SDO zal een Service Data Object (SDO) implementatie bieden, en de mogelijkheid scheppen om ieder willekeurig Java object als SDO te representeren.
- EclipseLink-DAS levert een SDO Data Access Service die een brug slaat tussen SDO en JPA.

- EclipseLink–DBWS biedt ondersteuning aan ontwikkelaars voor het snel en efficiënt ontsluiten van stored procedures, packages, tabellen en ad-hoc SQL statements in relationele databases als Web Services.
- EclipseLink-XR biedt ondersteuning voor situaties waar XML nodig is uit een relationele database, door de uitgebreide mogelijkheden van EclipseLink-ORM en EclipseLink-OXM te combineren.
- EclipseLink-EIS stelt ontwikkelaars in staat om Java POJOs te mappen naar niet-relationele data bronnen, gebruik makend van de Java Connector Architecture (JCA) API.

Hoewel dit project ondergebracht is in de Eclipse Foundation, kan het EclipseLink Persistence Platform in iedere Java omgeving gebruikt worden. Het is een runtime framework dat niet afhankelijk is van andere Eclipse componenten, noch van welke IDE, client of server omgeving dan ook. Wel zullen verschillende andere Eclipse Foundation projecten gebruik gaan maken van de faciliteiten die het EclipseLink Persistence Platform gaat bieden.
Met het doneren van de TopLink code aan dit project komt natuurlijk direct de vraag naar boven wat er nu gaat gebeuren met het product Oracle TopLink. Op het Oracle Technology Network is een FAQ opgenomen (http://www.oracle.com/technology/tech/eclipse/pdf/eclipselink-faq.pdf) die uitgebreid op deze en andere vragen ingaat. De belangrijkste punten hieruit zijn:

- Oracle TopLink blijft bestaan als product, inclusief ondersteuning door Oracle Support.
- Toekomstige versies van Oracle TopLink zullen op basis van de EclipseLink code gebouwd worden, en deze slechts uitbreiden om geavanceerde integratie met de Oracle Application Server en de Oracle SOA Suite mogelijk te maken.
- Oracle TopLink zal geen rijkere set features en mogelijkheden bieden dan EclipseLink.
- De Mapping Workbench maakt geen onderdeel uit van de initiële contributie aan het EclipseLink project. Navraag bij TopLink product management leert dat er wel plannen zijn om in de toekomst ook de design time code te doneren.

Conclusie

Oracle TopLink is een zeer krachtig ORM framework met een rijke historie. De laatste jaren heeft het product een aantal bijzondere ontwikkelingen doorgemaakt. Voor het eerst lijkt er sprake te zijn van een volledige commitment aan een standaard persistence API (EJB3 JPA) en is TopLink zelfs een referentie implementatie hiervan. Met het doneren van de runtime code van TopLink aan de open source community zal het gebruik van TopLink nog veel meer laagdrempelig worden. Door tenslotte de TopLink code te integreren met het EclipseLink Persistence Platform, dat een goede kans heeft om een de facto standaard persistence framework te worden, zal ook het gebruik van geavanceerde TopLink features die niet rechtstreeks door de JPA standaard ondersteund worden acceptabel zijn. Samen met alle andere EclipseLink componenten lijkt EclipseLink een zeer aantrekkelijke 'one-stop solution' te gaan bieden voor vrijwel alle persistence-gerelateerde aspecten van Java applicaties.

Resources

Informatie over Toplink geschiedenis:
http://www.oracle.com/technology/tech/java/newsletter/articles/toplink/history_of_toplink.html

Informatie over Toplink JPA:
http://www.oracle.com/technology/products/ias/toplink/jpa/resources/toplink-jpa-extensions.html
http://www.oracle.com/technology/products/ias/toplink/jpa/resources/faq.html

Informatie over EclipseLink:
http://otazi.blogspot.com/2007/03/what-is-eclipselink.html
http://www.oracle.com/technology/tech/eclipse/pdf/eclipselink-faq.pdf