

# Application Development Framework (ADF)

## Teil 3: Standardisierte Persistenzmechanismen in Verbindung mit ADF

Autor(en): Michael Bräuer, Kersten Mebus  
ORACLE Deutschland GmbH

*Das von Oracle zur Verfügung gestellte Framework ADF kann als eine Sammlung von verschiedenen Frameworks angesehen werden, wobei jedes dieser bereitgestellten Werkzeuge gewisse Teilprobleme löst, die bei der Anwendungsentwicklung in unterschiedlichen Projekten wiederkehren. So ermöglichen zum Beispiel die in unserem letzten Artikel (Teil 2 unserer Artikelserie) vorgestellten ADF Faces eine deklarative und durch Toolunterstützung sehr produktive Entwicklung von Web-basierten Anwendungen, u.a. durch Bereitstellung einer Reihe von vorgefertigten und wiederverwendbaren Bausteinen. Eine in fast jedem Java Projekt anzutreffende Herausforderung ist die Art und Weise, wie Objektstrukturen dauerhaft (persistent) gemacht werden, z.B. in relationalen Systemen. Mit Oracle TopLink steht hier seit mehr als 10 Jahren ein reifes Persistenzwerkzeug zur Verfügung. Es beinhaltet mit dem aktuellen Release 11g neben einer nativen Mappingbeschreibung und API auch eine Implementierung des Java Persistence API (JPA) namens EclipseLink JPA mit zusätzlichen, über das JPA 1.0 hinausgehenden Features. Im folgenden Artikel (Teil 3 unserer Artikelserie) soll auf verschiedene Persistenzierungsmöglichkeiten eingegangen werden, um im Anschluss an einem einfachen Code-Beispiel eine Einführung in JPA zu geben, EclipseLink und dessen JPA Spezialitäten vorzustellen und die Verwendung in Zusammenhang mit JDeveloper und ADF darzustellen.*

### **Die Herausforderung**

In einer Vielzahl von Projekten wird die Java Plattform als Basis für die Anwendungsentwicklung benutzt. Applikationen werden auf der Grundlage von objekt- und komponentenorientierten Prinzipien entworfen und implementiert. Objekte besitzen Eigenschaften und Verhalten und können zum Beispiel in Beziehungen untereinander stehen oder Verhalten und Eigenschaften von anderen Objekten erben. Zur Laufzeit entstehen komplexe Gebilde, die am Besten als

Objektgraphen betrachtet werden können. Dabei tritt eine immer wiederkehrende Anforderung auf: ein Teil der in der Programmlogik vorkommenden Objekte besitzen Eigenschaften, die dauerhaft in einer Datenquelle abgelegt und von dort gelesen werden müssen. Auf der anderen Seite haben sich über die letzten Jahrzehnte für die Datenhaltung relationale Systeme als am geeignetsten erwiesen. Daten werden in „flachen“ Tabellenstrukturen abgelegt. Auch hier können Beziehungen untereinander bestehen, Konzepte wie Vererbung oder Verhalten kennt die relationale Welt jedoch nicht (ähnliches gilt für das Ablegen und Lesen von Daten aus/in XML Strukturen oder Legacy Systemen). Die Integration beider Welten ist ein nicht zu unterschätzendes Problem, welches auch treffenderweise als „Object-Persistence Impedance Mismatch“ (siehe Abbildung 1) bezeichnet wird.

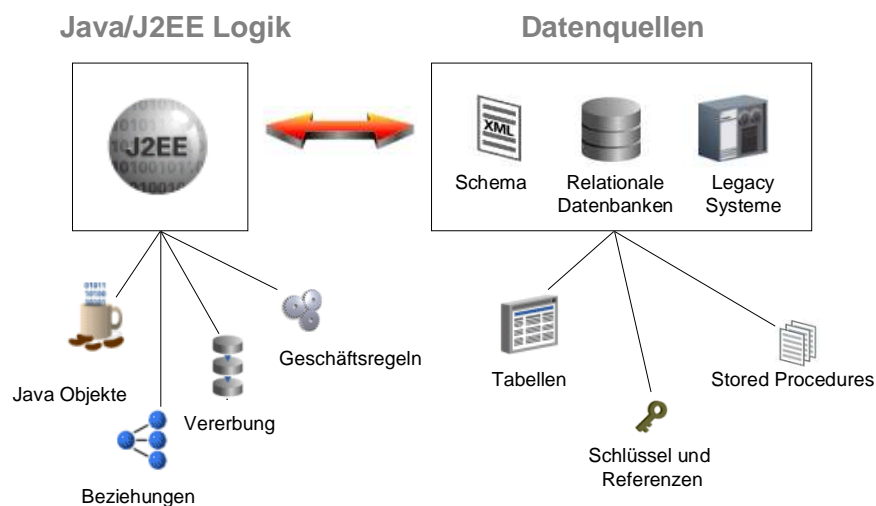


Abbildung 1: Unterschiedliche Konzepte müssen bei der Persistenzierung von Objekten beachtet werden.

Aus technischer Sicht müssen im objektrelationalen Fall im Wesentlichen folgende Aufgaben betrachtet werden:

1. Möglichkeit der Definition von flexiblen Mappings der unterschiedlichen Strukturen, möglichst ohne das ursprüngliche Objekt- bzw. Datenmodell anpassen zu müssen (Anforderung A1)
2. Bereitstellung effektiver und effizienter Schnittstellen zum Ablegen von Objekten in eine Datenbank und Lesen von Objekten aus der Datenbank in

einer objektorientierten Art und Weise (CRUD Operationen, Query/Expression Language, Transaktionsunterstützung, Caching etc.)

(Anforderung A2)

3. Bereitstellung einer Infrastruktur, die sich in unterschiedlichsten Umgebungen/Frameworks (J2SE, J2EE, ADF, Spring, OSGi etc.) integrieren lässt (Anforderung A3)

Die Herausforderung, die verschiedenen Konzepte unter einen Hut zu bringen, werden ausserdem dadurch verstärkt, dass verschiedene Projektbeteiligte (Java Entwickler, DBA) über verschiedene Fähigkeiten verfügen und sich oft nur für die Bereiche verantwortlich fühlen, in die sich die jeweiligen Personen spezialisiert haben. Es ergeben sich zwei interessante Fragen, die sich in diesem Zusammenhang stellen:

1. Wie kann Infrastrukturprogrammierung vermieden werden, damit der Entwickler sich der eigentlichen Arbeit, nämlich der Implementierung der Geschäftslogik widmen kann?
2. Kann eine Spezialisierung stattfinden, sodass sich einzelne Entwickler um das Thema Persistenz als Ganzes kümmern können? Gibt es Frameworks, in die sich einzelne Entwickler des Projektteams einarbeiten können und somit eine solche Spezialisierung gefördert wird?

### **Die Antwort: Persistenzierung mit Oracle Lösungen**

Oracle bietet in diesem Zusammenhang verschiedene Lösungen an, die die beiden obigen Fragen positiv beantworten (siehe Abbildung 2).

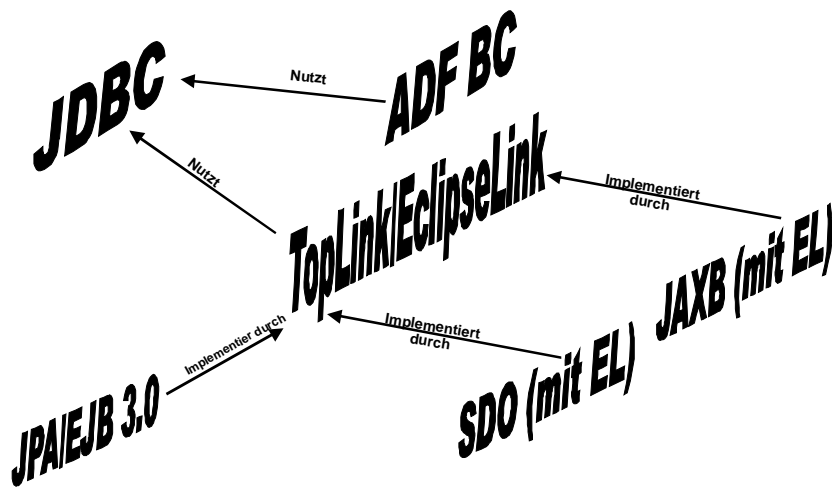


Abbildung 2: Verschiedene Frameworks: ADF Business Components (ADF BC), Oracle TopLink/EclipseLink (EL) und Java Persistence API (JPA).

Innerhalb des ADF Frameworks können sowohl die ADF Business Components als auch Oracle TopLink, hier zusammen mit EJBs, Java Klassen oder auch Web Services, dem Business Services Layer zugeordnet werden (siehe Abbildung 3).

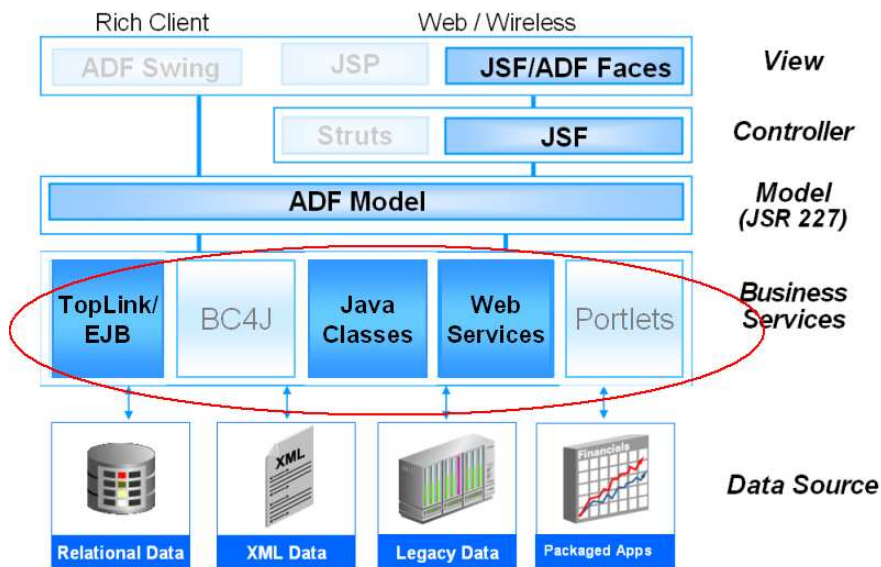


Abbildung 3: Oracle TopLink als Teil des Business Services Layer in Oracle ADF

Wie in unserem letzten Artikel beschrieben, sind die ADF BC besonders für die typische Situation von Forms-Kunden zugeschnitten: die Architektur ist festgelegt,

der Entwickler arbeitet sehr relationsnah und in Verbindung mit den ADF Datacontrols, Oracle ADF Faces und deren Databinding stellt sich die Verwendung dieser verschiedenen ADF Bausteine ähnlich der Verwendung einer 4GL Sprache dar. Im Gegensatz dazu bietet Oracle TopLink dem Java Programmierer die Flexibilität, ein gewünschtes Objektmodell auf ein bestehendes Datenmodell zu mappen (siehe Anforderung A1 oben), in einer objektorientierten Art und Weise unter Zuhilfenahme der Mappingdefinition mit der Datenbank zu kommunizieren (A2) und ausserdem innerhalb von verschiedenen Umgebungen und orthogonal mit anderen Frameworks zu arbeiten (A3). Ausserdem können mit TopLink neben relationalen Datenbanken auch andere Datenquellen und Standards adressiert werden. So setzen zum Beispiel die Implementierungen von SDO (Service Data Objects), JAXB (Java XML Binding) oder JPA (Java Persistence API) von Oracle auf Oracle TopLink auf. Die folgende Tabelle stellt die beiden Frameworks nochmals gegenüber:

Oracle ADF BC	Oracle TopLink
Generierung aus relationalen Strukturen	Objektorientierte Prinzipien wichtig – Domänenmodell steht im Vordergrund
Architektur vorgegeben (Entity Objekt, View Objekt, View Link, Association)	Entwickler gibt Objektmodell vor
Gut geeignet für 4GL Entwickler	Gut geeignet für OO Entwickler
Einsatz: datenbankbasierte Anwendungen	Einsatz: datenbankbasierte Anwendungsentwicklung, (Web-) Services Entwicklung, Java/XML Entwicklung
Datenquelle: relationale DB	Verschiedene Arten von Datenquellen und Standards
Einfaches O/R Mapping	Einfaches und komplexes O/R Mapping

### **EclipseLink - ein umfassendes Persistenzierungsframework**

Mit Oracle TopLink steht seit mehr als 10 Jahren ein reifes Persistenzwerkzeug zur Verfügung. Im Jahre 2007 entschied sich Oracle den Quellcode von Oracle TopLink 11g Preview 1 dem Eclipse Persistence Services Project (EclipseLink) zur Verfügung zu stellen ([1], [2]). Zuvor hatte Oracle mit TopLink Essentials schon die Referenzimplementierung für JPA 1.0 ([3]) als Open Source bereitgestellt ([4]).

TopLink Essentials wird unter anderem vom Oracle Application Server 10g (10.1.3.x) als Persistence Provider benutzt. Bei EclipseLink handelt es sich nicht nur um ein reines objektrelationales Mappingframework. EclipseLink setzt eine Reihe von weiteren Standards und Funktionalitäten um, u.a. (siehe hierzu auch Abbildung 4):

- Mapping von Objekt-XML Strukturen (MOXy) wie z.B. Unterstützung von JAXB 2 (Java Architecture for XML Binding)
- Unterstützung von SDO (Service Data Objects)
- Mapping auf Legacy Systeme (EIS)
- Datenbank Web Services (DBWS) (geplant)

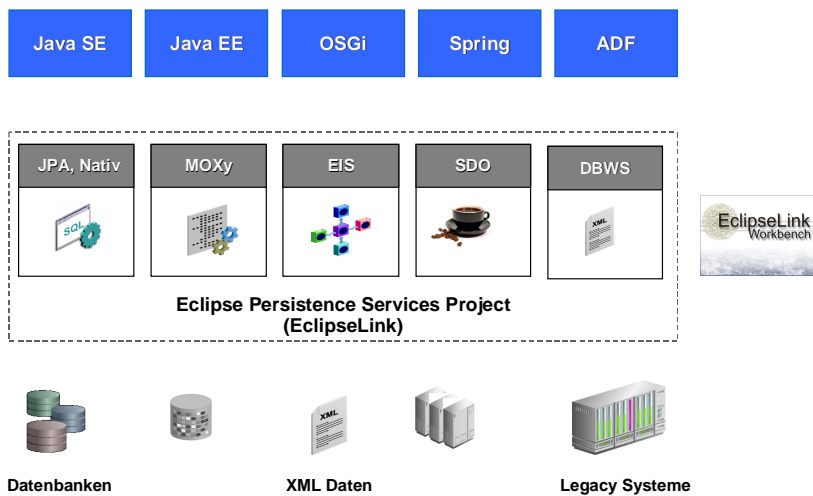


Abbildung 4: Umfang von EclipseLink

Seit Oktober 2008 ist Oracle TopLink 11g verfügbar. Hier schliesst sich wieder der Kreis: Oracle TopLink 11g enthält EclipseLink 1.0.1, so dass Kunden durch Erwerb von TopLink 11g Lizenzen und einem Supportvertrag bei Bedarf Support für EclipseLink durch Oracle Support Services in Anspruch nehmen können ([5]).

Eine besondere Stellung für die Umsetzung von objektrelationalen Mappings in einer standardbasierten Art und Weise nimmt das Java Persistence API ein. EclipseLink unterstützt JPA 1.0 (JSR-220) und wird die JPA 2.0 (JSR 317, siehe [6]) Referenzimplementierung umsetzen. Der aktuelle Entwicklungsstatus hierzu kann unter [7] verfolgt werden.

Das Java Persistence API ist ein leichgewichtiges Framework, welches auf POJOs (Plain Old Java Objects) beruht. Es liefert einen portablen und standardbasierten Ansatz, bei dem die Zuordnung zwischen persistenzierbaren Java Objekten (Entitäten) und relationalen Strukturen deklarativ mittels Annotation oder durch Angabe von Informationen in bestimmten XML Dateien erfolgt. Zudem stellt es eine Reihe von Interfaces und eine SQL ähnliche Abfragesprache (Java Persistence Query Language – JP QL) bereit, welche benutzt werden können, um Objekte abzufragen, zu erzeugen, zu verändern oder zu löschen. Dies kann sowohl in einer J2SE 5.0 (oder höher) als auch EJB 3.0 konformen J2EE Umgebung geschehen. Zur Illustrierung ein einfaches Beispiel:

```
import javax.annotations.*; ...
/*
 * Persistenzierbares POJO
 */
@Entity
public class Kunde implements Serializable {
    @Id
    @Column(name="KUNDEN_NR")
    private Long id; //Persistenzidentität

    //es gibt eine Spalte vorname in der Tabelle Angestellter
    private String vorname;

    @OneToOne //Mapping einer Beziehung
    @JoinColumn(name="ADDR_ID",
        referencedColumnName = "ADDRESS_ID")
    private Adresse adresse;

    ...
}
```

Die Angabe von Mappinginformationen erfolgt nach dem Prinzip „Konfiguration nur in der Ausnahme“. Zum Beispiel wurde für die Entität keine explizite Zuordnung zu einer Tabelle durchgeführt – es wird zur Laufzeit die Existenz einer gleichnamigen Tabelle unterstellt.

Die Paketierung der einzelnen Entitäten erfolgt in einer speziellen Datei mit dem Namen `persistence.xml`. Hier sind zusätzliche Persistence Provider spezifische Konfigurationseigenschaften anzugeben wie z.B. Log Level, Cache Verhalten u.ä.

Als Persistence Provider bezeichnet man eine spezielle JPA Implementierung – so sind z.B. EclipseLink JPA oder auch TopLink Essentials zwei am Markt befindliche Persistence Provider. Oracle's strategischer Persistence Provider ist EclipseLink JPA.

Wie werden die Mapping Informationen nun programmatisch benutzt? Innerhalb der Anwendung wird das Interface `EntityManager` benutzt, um Operationen auf und mit Entitäten durchzuführen. Dieses Interface ist der zentrale Einstiegspunkt. Wie eine Implementierung dieses Interfaces benutzt wird, hängt von der benutzten Umgebung ab. Das folgende Beispiel zeigt die Verwendung einer `EntityManagerFactory` in einer Java SE Umgebung. In EJB 3.0 konformen JEE Umgebungen kann eine `EntityFactory` auch mittels JNDI Lookup oder Injection vom Container bereitgestellt werden.

```
...
/*
 * Methode zum Speichern eines Kunden
 */
public void persist(EntityManagerFactory emf, Kunde kunde) {
    EntityManager em = emf.createEntityManager();
    try {
        em.getTransaction().begin();
        em.persist(kunde); //Kunden speichern
        em.getTransaction().commit();
    } catch (Exception ex) {
        if (em.getTransaction().isActive())
            em.getTransaction().rollback();
        // normalerweise nun Exception propagieren
    } finally {
        em.close();
    }
}
/*
```

```

* Methode zum Abfragen aller Kunden
*/
public void List<Kunde> queryAll(EntityManagerFactory emf) {
    EntityManager em = emf.createEntityManager();
    Query allQuery = null;
    List<kunde> kundenListe= null;
    try { //alle Kunden holen
        allQuery = em. createQuery("SELECT k FROM Kunde k");
        kundenListe = allQuery.getResultList();
    } finally {
        em.close();
    }
} ...

```

Wie sieht es mit Unterstützung von Transaktionen aus? Normalerweise werden Veränderungen von Eigenschaften einer Entität, das Erzeugen und Löschen einer Entität innerhalb einer Transaktion ausgeführt. Dies spiegelt sich dann darin wieder, das entsprechende Veränderungen innerhalb der Datenbank entweder komplett erfolgreich ausgeführt werden oder als eine ganze Einheit scheitern. Im Speicher (JVM) können jedoch Entitäten verändert werden, ohne das der veränderte Zustand der Entitäten dauerhaft gemacht wird. Entitäten sind somit quasi-transaktional. Im J2EE Container stehen als Transaktionen JTA basierte Transaktionen zur Verfügung, im J2SE Fall kann man wie im obigen Beispiel den `EntityManager` bemühen, um innerhalb einer Transaktion zu arbeiten.

### **EclipseLink JPA Extensions**

Neben den von der JPA 1.0 vorgesehenen Funktionalität bietet EclipseLink u.a. innerhalb der sogenannten EclipseLink JPA Extensions weitergehende Eigenschaften an. Diese umfassen u.a.:

- Zusätzliche Mappings (Converter, Transformation, BasicCollection etc.)
- Unterstützung von Stored Procedures
- Ausgefeilte Cacheunterstützung/-konfiguration
- Zusätzliche Policies für Locking
- Hints für optimierte Zugriffe (Batch Reading, Joined Reading)
- Individuelle Anpassungen, z.B. durch Customizer

Eine ausführliche Diskussion dieser Features würde den Rahmen dieses Artikel sprengen. Es wird auf [8] und [9] verwiesen.

## Toolunterstützung und Verwendung in ADF

Der JDeveloper bietet eine umfangreiche JPA Unterstützung an. Schon im JDeveloper 10g gab es die Möglichkeit, Entitäten anhand von Tabelleninformationen zu generieren oder Artefakte nativer TopLink Projekte graphisch durch Wizards zu generieren und zu bearbeiten.

Der JDeveloper 11g geht hier bei der Unterstützung von JPA weiter. Er liefert umfangreiche, graphische Editoren zum Anlegen und Ändern von Artefakten, die für die Nutzung des JPA notwendig sind. Dieses umfasst zum Beispiel einen graphischen Editor für die Konfigurationsdatei `persistence.xml`. Hier kann visuell das Mapping durchgeführt werden, ähnlich wie es von der TopLink/EclipseLink Workbench angeboten wird. Zusätzlich sind auch im Property Inspector die editierbaren Informationen zu den JPA Mappings zu finden.

JPA Entities können durch EJB Session/Java Fassaden innerhalb von ADF benutzt werden. Dies kann per Hand oder per Wizard geschehen (Abbildungen 5 und 6). Auf Basis der Fassaden können zugehörige DataControls generiert werden, die die Grundlage für die grafische Oberflächengestaltung in Form von zum Beispiel ADF Faces bilden [10,11]

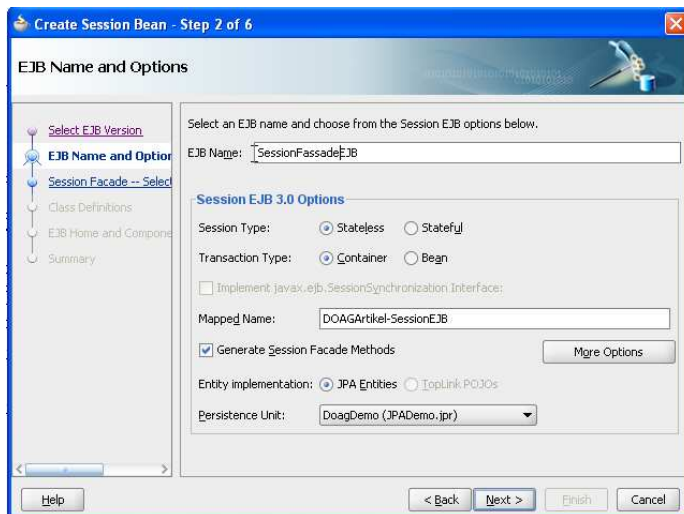


Abbildung 5: Generierung einer EJB Session Bean im JDeveloper 11g

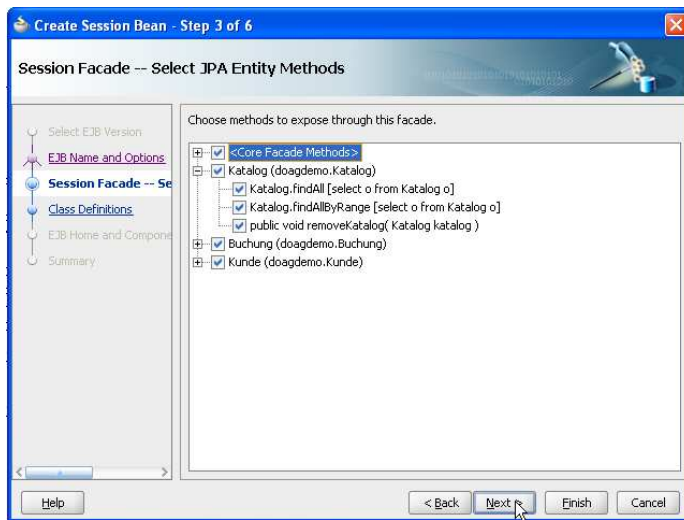


Abbildung 6: Auswahl der Methoden der Entitäten, die durch die Bean nach aussen zur Verfügung gestellt werden sollen

Um einen ersten Test auszuführen, kann ein EJB Client im JDeveloper generiert werden. Anschließend wird die Bean und der Client auf dem mit dem JDeveloper 11g mitgelieferten und integrierten Oracle WebLogic 10.3 Server ausgeführt indem man beide Ziele startet (Run).

## Zusammenfassung

Mit EclipseLink/Oracle TopLink 11g steht ein mächtiges Persistenzwerkzeug für Java Projekte zur Verfügung. Besonders interessant im Zusammenhang mit Objekt-relationalen Mappinganforderungen ist das JPA. Es stellt ein leichtgewichtiges standardisiertes Framework dar, welches von TopLink aktuell in der Version 1.0 unterstützt wird. Zusätzlich werden dem Entwickler vielfältige, über die Spezifikation hinausgehende Features durch die EclipseLink JPA Extensions an die Hand gegeben. Mit dem JDeveloper 11g steht zusätzlich ein Tool zur Verfügung, welches umfangreiche Unterstützung für Persistenzprojekte bietet.

Durch den Einsatz dieser Werkzeuge kann Infrastrukturprogrammierung vermieden werden, damit sich Entwickler und Entwicklerinnen der eigentlichen Arbeit, nämlich der Implementierung der Geschäftslogik widmen können. Zudem kann eine Spezialisierung stattfinden, sodass sich einzelne Entwickler und Entwicklerinnen um das Thema Persistenz kümmern können.

## Nützliche Links

- [1] Press Release: Oracle Proposes Open Source Persistence Project at Eclipse Foundation  
[http://www.oracle.com/corporate/press/2007\\_mar/OpenSource-TopLink.html](http://www.oracle.com/corporate/press/2007_mar/OpenSource-TopLink.html)
- [2] Eclipse Persistence Services Project  
[http://www.eclipse.org/projects/project\\_summary.php?projectid=rt.eclipselink](http://www.eclipse.org/projects/project_summary.php?projectid=rt.eclipselink)  
<http://www.eclipse.org/eclipselink>
- [3] JSR 220: Enterprise JavaBeans™ 3.0, beinhaltet JPA 1.0  
<http://jcp.org/en/jsr/detail?id=220>
- [4] Press Release: Oracle Extends Java Leadership; Unveils Reference Implementation for Java Persistence API in Open Source  
[http://www.oracle.com/corporate/press/2006\\_may/toplinkessentials-ri\\_0.html](http://www.oracle.com/corporate/press/2006_may/toplinkessentials-ri_0.html)
- [5] Oracle Releases Oracle® TopLink® 11g Based on EclipseLink  
[http://www.oracle.com/us/corporate/press/017498\\_EN](http://www.oracle.com/us/corporate/press/017498_EN)
- [6] JSR 317: Java™ Persistence 2.0  
<http://jcp.org/en/jsr/detail?id=317>
- [7] Statusseite Umsetzung EclipseLink JPA 2.0 Unterstützung  
[http://wiki.eclipse.org/EclipseLink/Development/JPA\\_2.0](http://wiki.eclipse.org/EclipseLink/Development/JPA_2.0)
- [8] Clarke, D., Smith, S.: Introducing EclipseLink  
<http://java.dzone.com/articles/introducing-eclipselink>
- [9] Using EclipseLink JPA Extensions (ELUG)  
[http://wiki.eclipse.org/Using\\_EclipseLink\\_JPA\\_Extensions\\_%28ELUG%29](http://wiki.eclipse.org/Using_EclipseLink_JPA_Extensions_%28ELUG%29)
- [10] DOAG News Q3/2008  
Application Development Framework (ADF) Teil 1: Überblick
- [11] DOAG News Q4/2008  
ADF Teil 2: Der Einstieg in die J2EE-Welt für Forms-Entwickler

### Kontaktadressen:

Michael Bräuer

[michael.braeuer@oracle.com](mailto:michael.braeuer@oracle.com)

Kersten Mebus

[kersten.mebus@oracle.com](mailto:kersten.mebus@oracle.com)

Jürgen Menge

[juergen.menge@oracle.com](mailto:juergen.menge@oracle.com)

Detlef Müller

[detlef.mueller@oracle.com](mailto:detlef.mueller@oracle.com)