# Teilaspekte einer Migration von Forms nach ADF

**Manuel Malatyali**
**TEAM GmbH**
**Paderborn**

**Schlüsselworte**
ADF, Migration Forms nach ADF

## Einleitung

Ein technologisches System ist solange aktuell, wie es den Anforderungen genügen kann. Im Laufe der Zeit wachsen die Anforderungen, so dass die Funktionalitäten des Systems ebenfalls wachsen müssen. Mit Alter des Systems wächst der Aufwand neue Funktionalitäten zu entwickeln, da diese ursprünglich nicht im Systemdesign berücksichtigt wurden. Oracle Forms ist ein solches System, das den heutigen Anforderungen nur noch mit großem Aufwand genügt bzw. viele Erwartungen schlicht nicht erfüllen kann. Neben der Motivation auf ein neues System zu migrieren stellt sich die Frage, wie der PL/SQL-Code migriert wird. Dieser Vortrag soll aufzeigen, welchen Pfad eine Migration nach ADF nehmen kann.

## Die Zukunft von Forms ist ~~Forms~~ ADF

Eine Oracle Forms Anwendung hat ein datennahes Arbeiten im Fokus. Das heißt die Oberfläche ist stark an das Datenbankschema angelehnt. Folglich arbeitet ein Benutzer mit mehreren verschiedenen Dialogen, um eine Arbeitseinheit (Use Case) auszuführen. Er öffnet verschiedene Dialoge, um alle benötigten Informationen im Blick zu haben und verwendet dieselben oder wiederum andere Dialoge, um Änderungen durchzuführen. Der Aufbau der Dialoge orientiert sich an der Struktur der (Datenbank-) Tabellen und Views – nicht an die Bedürfnisse des Benutzers. Eine Motivation für eine Migration besteht darin diese Nutzungserfahrung (User Experience) signifikant zu verbessern. Wünschenswert ist eine Oberfläche, die den Bedürfnissen des Nutzers entsprechend aufgebaut ist und ihm bei der Abarbeitung eines Arbeitsschrittes führend unterstützt.

Im Rahmen eines Migrationsprojektes stellt sich neben den Oberflächen die Frage, wie mit der Geschäftslogik umgegangen wird, die in dem Forms-System vorhanden ist. Innerhalb dieses Vortrages ist die Voraussetzung gegeben, einen möglichst großen Teil der bestehenden PL/SQL-Investition zu bewahren. Die Ausführung von PL/SQL in der Datenbank verstößt per Definition gegen das Designprinzip einer dreischichtigen Architektur, die die Geschäftslogik innerhalb der Mittelschicht ausführt und die Datenbank lediglich als „Data-Store“ sieht. Es werden die daraus folgenden Problemstellungen beleuchtet, die bei dem Aufruf einer Funktion anfangen und bei den Konfigurationsparametern einer entsprechenden Umgebung aufhören.

Oracle ADF bietet keine deklarative Integration von PL/SQL. Trotzdem ist es leichter eine PL/SQL-freundliche Umgebung zu schaffen, als mit vergleichbaren Frameworks. Dies wird deutlich anhand der Möglichkeiten, z.B. optimistisches Locking oder einen für den Benutzer reservierten Application Module-Kontext zu konfigurieren, statt dieses neu konzipieren und ggf. durch Java neu implementieren oder anpassen zu müssen.

**Forms „under the hood“**

Betrachtet man eine Forms Anwendung im Detail, so wird ersichtlich, dass PL/SQL-Code nicht gleich PL/SQL-Code ist. Somit ist auch die Frage, ob PL/SQL-Code in die Datenbank verlagert werden soll, nicht mit einem einfachen ja oder nein zu beantworten. In Forms wird PL/SQL sowohl für die Implementierung von technischen Aspekten, als auch zur Realisierung von fachlichen Aspekten verwendet. Unter technischen Aspekten fallen z.B. Querschnittsfunktionalitäten (Fehlerbehandlung, Mehrsprachigkeit, Autorisierung/ Authentifizierung, Tracing / Logging, etc.) oder Funktionalitäten, um einen Forms-Wechsel zu kapseln (OPEN_FORM, NEW_FORM, etc.).
Zu den fachlichen Aspekten zählen u.a. UI- und Geschäftslogik. In diesem Projekt wurde sämtliche Geschäftslogik, die keinen direkten Bezug zur Oberfläche hat, in PL/SQL-Packages in der Datenbank geschrieben, was den Aufwand einer Migration an dieser Stelle drastisch sinken lässt.

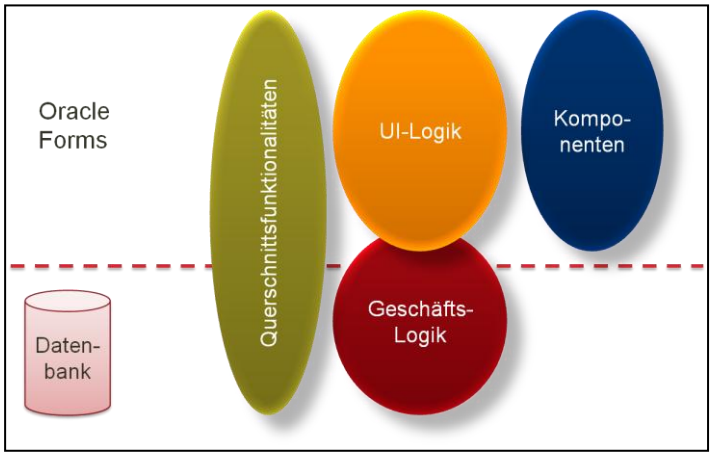


*Abb. 1: (Teil-) Aspekte der PL/SQL-Nutzung bei Oracle Forms.*

Hier zeigt sich zum Einen das Potenzial durch geeignetere Werkzeuge als PL/SQL eine Verbesserung zu schaffen (z.B. explizite graphische Darstellung des Workflows). Zum Anderen liegt hier die Schwierigkeit einer Migration des PL/SQL-Codes, nämlich den bestehenden Code zu klassifizieren und eine bestmögliche Abbildungen zu finden. Ein Auszug aus den Optionen die Aspekte abzubilden zeigt die folgende Tabelle:

| Aspekt in PL/SQL | Migrationsziel |
|---|---|
| Form-Wechsel (CALL_FORM, etc) | Prozessdarstellung |
| Querschnittsfunktionalitäten | Datenbankseitige Implementierung / Tabellen übernehmen, Reimplementierung der Forms-Logik |
| UI-Logik | komplette Reimplementierung |
| Geschäftslogik | Beibehalten / Sanieren |

Im Rahmen des Projektes ist aufgefallen, dass sich die Realisierung von Querschnittsfunktionalitäten (QF) in einem ADF-Projekt „auf der grünen Wiese“ stark unterscheidet zu einem ADF-Migrationsprojekt. Bereits bestehende Konzepte (z.B. Nutzerverwaltung) werden über die Datenbank abgewickelt und sollen weiterhin verwendet werden. Die Reimplementierung in ADF beschränkt sich dabei nicht nur auf Code in einige Basisklassen, die transparent im Hintergrund verwendet werden, sondern zieht sich von den angesprochenen Basisklassen über Task Flow Templates bis hin zu Page Templates und entsprechenden Vorgehensmodellen einen Dialog in ADF zu erstellen. Bei der Integration der QF hat sich gezeigt, dass die einzelnen ADF Komponenten gut aufeinander abgestimmt sind, denn gerade bei dem Thema Wiederverwendbarkeit wären bei einem schlechten Design große Probleme aufgetreten.

### ADF in einem Migrationsprojekt

Nach dem Postulat Geschäftslogik während einer Migration „nur“ zu sanieren bzw. beizubehalten stellt sich die Frage, wie eine PL/SQL-Funktion in ADF eingebunden werden kann, sodass z.B. bei einem Buttonclick diese Funktion mit entsprechenden Parametern gefüllt und ausgeführt wird. Den Handlungsbedarf soll folgender Vergleich einer Forms Implementierung zu einer ADF-Implementierung eines PL/SQL-Funktionsaufrufs zeigen:

**Forms**

```
-- T_BUTTON_PRESSED
begin
 pk_doag.anmelden (
    :bl_dialog.vorname,
    :bl_dialog.nachname,
    :bl_dialog.firma,
    :bl_dialog.bemerkung,
    :bl_dialog.bestaetigung
 )
end;
```

**ADF (Java)**

```
public void callPkDoagAnmelden() throws SQLException {

  DoagViewRowImpl row = (DoagViewRowImpl) getCurrentRow();

  String vorname = row.getVorname();
  String nachname = row.getNachname();
  String firma = row.getFirma();
  String bemerkung = row.getBemerkung();
  Number bestaetigung = null;

  DBTransaction trans = getDBTransaction();
  String stmnt = "begin pk_doag.anmelden(?,?,?,?,?); end;";

  OraclePreparedStatement ops = (OraclePreparedStatement)
                trans.createPreparedStatement(statement,
                DBTransaction.DEFAULT);

  ops.setString(1, vorname);
  ops.setString(2, nachname);
  ops.setString(3, firma);
  ops.setString(4, bemerkung);
  ops.registerOutParameter(5, OracleTypes.NUMBER);

  bestaetigung = new Number(ocs.getNUMBER(5));
  ops.execute();

  row.setBestaetigung(bestaetigung);

}
```

Der direkte Vergleich zwischen der Einbindung in Forms und in Java zeigt, dass die Verwendung der JDBC-Schnittstelle den Aufwand, eine Funktion in der Datenbank aufzurufen, drastisch steigen lässt. Neben dem höheren Aufwand eine Funktion aufzurufen kommt erschwerend hinzu, dass bei einer Änderung der Funktion **kein** Fehler und keine Warnung im JDeveloper zur Designzeit angezeigt wird, somit ein Fehler erst zur Laufzeit identifiziert wird (oder durch entsprechende Test-Fälle).
An dieser Stelle verwenden wir eine eigens entwickelte JDeveloper Extension, die an die ADF-Business Components andockt und hier dem Entwickler die Möglichkeit bietet PL/SQL-Code zu schreiben, ohne sich dabei um den Verwaltungsaufwand zu kümmern.

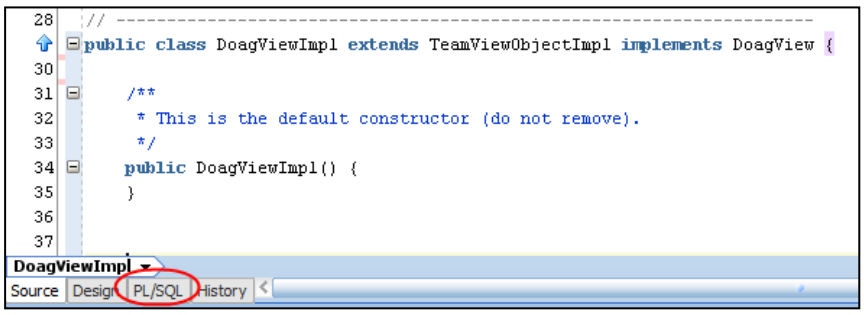


*Abb. 2: JDeveloper-Extension für PL/SQL*

**Komponentenentwicklung mit ADF**

Nachdem die Einbindung von Querschnittsfunktionalitäten und der Geschäftslogik gesichert ist, wird schnell deutlich, dass eine automatische Migration der „in Stein gemeißelten“ Forms-Dialoge auf ähnliche ADF-Dialoge nicht sinnvoll ist, um eine Verbesserung der Nutzungserfahrung sicher zu stellen. Es ist klar, dass es nicht **eine** Anwendung für **jeden** Benutzer in **jeder** Umgebung gibt, sondern die beste Nutzungserfahrung entsteht, wenn die Anwendung auf den Benutzer, seinen Erwartungen und seiner Umgebung entsprechend angepasst ist. Dies beinhaltet insbesondere einen Bezug zu seinem Workflow und der Anzeige der Daten, die er „im Blick“ haben muss. Somit ist die Anforderung entstanden mithilfe von ADF, kleine überschaubare, Komponenten zu entwickeln und anschließend auf einer höheren Ebene zu einer neuen Komponente zusammenzustecken. In dem von TEAM entwickelten Framework existieren zwei Abstraktionsstufen, die von der ersten zur zweiten Stufe einzelne atomare Bausteine zu einer Oberfläche zusammensteckt und von der zweiten zur dritten Stufe den Workflow des Benutzers abbildet und ihn bei seiner Tätigkeit führt.

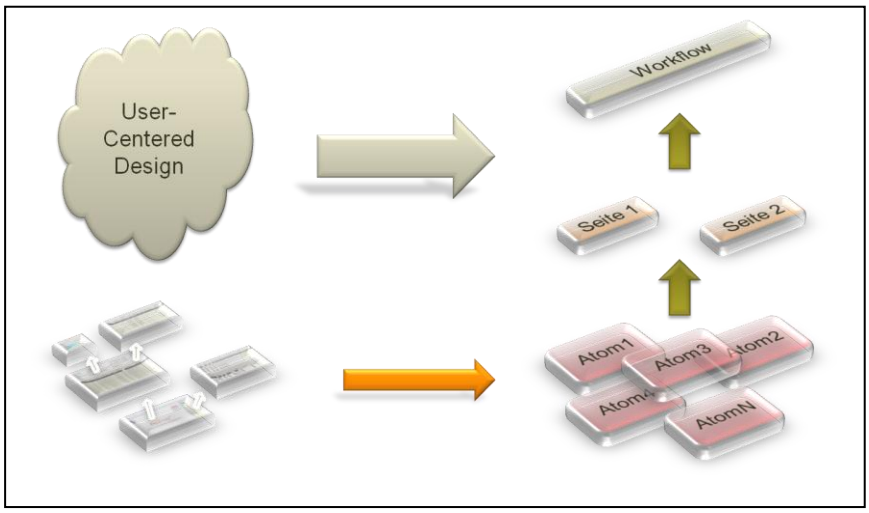


*Abb. 3: Schematische Darstellung*

Rückwirkend heißt dies für den Migrationsprozess, dass zunächst aus den Forms-Dialogen Teilkomponenten identifiziert (Abbildung 3, links unten) und auf atomare ADF-Komponenten abgebildet werden. Zu der Identifizierung gehört eine Analyse der Abhängigkeiten, sodass bei Dialogen, die sehr viele Abhängigkeiten besitzen, sich eine Sanierung vor einer Migration als sinnvoll herausgestellt hat. Die Unterteilung in kleinere Komponenten und die Trennung zwischen Komponentendefinition und -verwendung ermöglicht nicht nur Oberflächen leichter zusammenzustecken, sondern verbessert die Wartbarkeit und verbessert die Produktentwicklung durch die Vergabe von Verantwortlichkeiten.

**TEAM ADF Framework**

Es sind verschiedene Aspekte, die bei einer Migration eine Rolle spielen genannt worden. Doch wie fügen sie sich zu einem Gesamtbild zusammen? Die aus dem Projekt gewonnene Erfahrung wurde konsolidiert und ein Konzept erarbeitet, welches sämtliche Aspekte in einem Framework vereint (TEAM ADF Framework). Dieses Framework beschränkt sich dabei nicht auf den Anwendungsfall einer Migration, sondern wird ebenfalls für eine Neuentwicklung mit ADF eingesetzt. Neben Java-Klassen und Templates gehört ein definiertes Vorgehensmodell zu dem Framework, sodass durch Wiederhol- und Nachvollziehbarkeit eine gesicherte Migration ermöglicht wird (vgl. Auszug Abb. 4).

Die Realisierung der Migration erfolgt innerhalb dieses Migrationsprojektes von Hand. Zielsetzung für ein parallel laufendes Projekt ist es, aufwändige oder fehleranfällige Aufgaben zu identifizieren und durch ein Werkzeug automatisiert abarbeiten zu lassen.

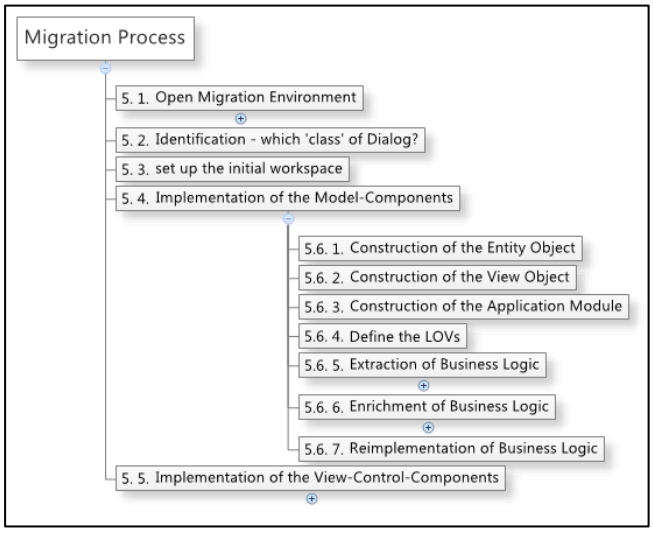


*Abb. 4: Auszug aus Vorgehensmodell: Migration eines atomaren Bausteins.*

**Kontaktadresse:**
Manuel Malatyali
TEAM GmbH
Partner für Technologie und
angewandte Methoden der
Informationsverarbeitung GmbH
Hermann-Löns-Straße 88
D-33104 Paderborn

Telefon: +49 (0) 5254 / 8008 – 43
Fax: +49 (0) 5254 / 8008 – 19
E-Mail: mma@team-pb.de
Internet: www.team-pb.de