Oracle ADF - bestehende PL/SQL-Logik ersetzen oder wiederverwenden ?

Jürgen Menge
Oracle Deutschland B.V. & Co. KG
München

Schlüsselworte: PL/SQL, Geschäftslogik, Java, Oracle Forms, Oracle ADF

Einleitung

In vielen Applikationen ist ein beträchtlicher Teil der Geschäftslogik in PL/SQL innerhalb der Datenbank implementiert. Soll nun eine Modernisierung der Applikationen in Richtung Java/ADF erfolgen, stellt sich die Frage, was mit der existierenden Logik in der Datenbank geschehen soll. Eine mögliche Vorgehensweise wäre die komplette Neu-Implementierung in Java innerhalb der Mittelschicht. Dem stehen in der Praxis aber folgende Punkte entgegen:

- in der vorhandenen PL/SQL-Logik steckt ein erhebliches Investment
- datennahe Operationen können innerhalb der Datenbank performant ausgeführt werden

Zudem stellt sich die Frage nach dem Nebeneinander von alten (z.B. Forms-) und neuen (Java/ADF-) Applikationen, die nach Möglichkeit dieselbe Geschäftslogik benutzen sollen. Die gemeinsame Nutzung einer Geschäftslogik-Schicht in der Datenbank kann damit Teil einer längerfristigen Migrations-Strategie sein.
Viele Kunden bevorzugen aus diesen Gründen einen pragmatischen Ansatz, der die weitere Nutzung von vorhandenen PL/SQL-Programmeinheiten einschließt, ja diese zum Teil sogar erweitert.

Integration von PL/SQL in Java/Oracle ADF

Generell gehört der Zugriff auf PL/SQL nicht zum Standard-Repertoir verbreiteter Persistenz-Frameworks. Auch Oracle ADF bietet aktuell keine deklarative Unterstützung, wie man sie z.B. von prozedur-basierten Blöcken in Oracle Forms kennt.
Welche Möglichkeiten bietet nun das Framework Oracle ADF, speziell die ADF Business Components (ADFbc), um diese Art des Zugriffs zu realisieren?
Im folgenden sollen zwei Fälle betrachtet werden, die in der Praxis häufig miteinander kombiniert werden. Zu jedem dieser Fälle werden mögliche Implementierungsvarianten aufgeführt.

(1) Datenzugriffe (Select, Insert, Update, Delete) über Table API bzw. Data Access Layer

1. Lesender Zugriff über View Objects auf Basis von Datenbank-Views;
   Überschreiben der Methode doDML() für die Entity Objects (Class *EntityImpl*)
   [http://www.oracle.com/technetwork/articles/muir-designer-085584.html]
2. Lesender Zugriff über View Objects auf Basis von Datenbank-Views;
   Verwenden von INSTEAD-OF-Triggern in der Datenbank
3. Kapselung der PL/SQL-Logik als Web Services innerhalb der Datenbank (SOAP)
   [http://sql-plsql-de.blogspot.co.uk/2012/05/soap-zugang-zu-plsql-database-native.html]
   oder über den APEX Listener (REST)
   [http://download.oracle.com/otn/java/appexpress/1.1/docs/AELIG/E21058_01.pdf
   (Kapitel 3)];
   Aufruf der Web Services über ADF Data Controls oder Web Service Proxy

(2) Geschäftslogik (Validierungen, Berechnungen, Ableitungen)

1. Eigene Methoden in der Implementierungsklasse des Application Module schreiben; Aufruf der PL/SQL-Routinen über Callable Statements (Funktionen) oder Pepared Statements (Prozeduren);
   Registrieren der Methoden als Client Interface
   [Duncan Mills, Peter Koletzke and Avrom Roy-Faderman, Oracle JDeveloper 11g Handbook: A Guide to Fusion Web Development (Oracle Press), 2009]
   Oracle JDeveloper 11g Handbook: A Guide to Fusion Web Development (Oracle Press)
2. Kapselung der PL/SQL-Logik als Web Services innerhalb der Datenbank (SOAP) oder über den APEX Listener (REST) [siehe oben]

Die generelle Schwierigkeit besteht in der Abbildung von Konstrukten aus PL/SQL in Java. So sind beispielsweise folgende Einschränkungen in Java bzw. JDBC zu beachten:

- globale Package-Variablen sind bei Connection Pooling nicht verwendbar
- kein Datentyp Boolean in Java
- keine Unterstützung für *%rowtype* bzw. *table of %rowtype*
- keine Unterstützung für Record Types

Grundsätzlich sollte bei einer Integration von PL/SQL-Logik diese durch eine Zwischenschicht (Wrapper) gekapselt werden. Durch diese Art der losen Kopplung ist ein späterer Austausch der Geschäftslogik möglich, ohne dass die darüberliegende Applikation davon betroffen ist.
Eine elegante Möglichkeit besteht z.B. darin, für die *%rowtypes* Object Types in der Datenbank zu generieren. Diese werden durch die Verwendung der Klasse *oracle.sql.Struct* in JDBC unterstützt. Werden der Implementierungsklasse des Application Modules eigene Interface-Methoden hinzugefügt, empfiehlt es sich, dafür ein spezielles Application Module zu verwenden, von dem die Application Modules der Applikation abgeleitet werden [http://www.youtube.com/watch?v=gOUUllRjDpU].

Aus den bisherigen Darlegungen ergibt sich folgende Architektur, die eine parallele Nutzung der PL/SQL-Geschäftslogik sowohl durch Oracle Forms- als auch durch Java/ADF-Applikationen ermöglicht:

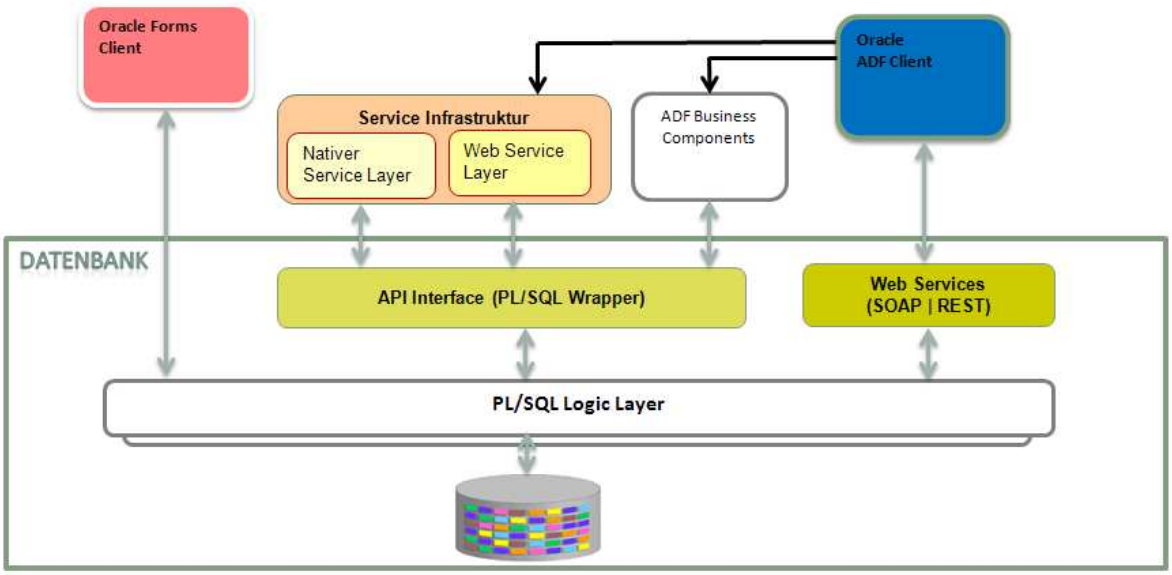

Abb. 1: Integration von PL/SQL-Logik in eine Anwendungs-Architektur

Eine der größten Herausforderung besteht darin, Änderungen an den Datenstrukturen bzw. in den Signaturen der Logik über alle Schichten korrekt und effizient nachzuziehen, zumal durch die Frameworks bzw. IDEs (Oracle JDeveloper) keine dedizierte Unterstützung dafür angeboten wird.
Als mögliche Vorgehensweisen kommen hier in Betracht:

- ein generischer Ansatz, d.h. eine universelle Schnittstelle zum Aufruf des PL/SQL [z.B. http://mohammedsaadadf.blogspot.de/2012/05/executing-stored-procedure-and.html]
- ein generativer Ansatz, d.h. die automatisierte Generierung aller Komponenten bei Änderungen [z.B. http://www.youtube.com/user/ADFTalk?feature=watch]

Kontaktadresse:

Name
Oracle Deutschland B.V. & Co. KG
Riesstr. 25
D-80992 München

Telefon: +49 (0) 89-1430 2239
Fax: +49 (0) 89-1430 2150
E-Mail juergen.menge@oracle.com
Internet: www.oracle.com