Der Tiger im Tank:
PL/SQL-Logik in Java-Anwendungen optimal nutzen

Thomas Haskes & Jens Hüttemann
Triestram & Partner GmbH (t&p)
Bochum

Schlüsselworte

Integration von datenbank-seitigem PL/SQL in eine JEE / mehrschichtige Java-Architektur mittels EclipseLink, Migration einer Oracle-Forms-Anwendung in Richtung Java und Open Source

Einleitung

Business-Anwendungen auf Basis von Oracle Forms enthalten mitunter einen beachtlichen Anteil von PL/SQL in der Oracle Datenbank. Eine solche Architektur, in der Business-Logik in der Datenbank implementiert ist, wird auch als datenbank-zentriert bezeichnet. Im Unterschied dazu ist die Architektur von webbasierten Java-Enterprise-Anwendungen meist mittelschicht-zentriert, da hier die Business-Logik im Application Server implementiert wird. Schließen sich diese beiden Architektur-Ansätze / Paradigmen gegenseitig aus? Und bedeutet es, dass bei der Migration einer datenbank-zentrierten Oracle Forms Anwendung die vorhandene PL/SQL-Logik in der Datenbank komplett zu verwerfen ist und im Application Server neu implementiert werden muss? Diese und ähnliche Fragen beschäftigen viele IT-Verantwortliche, die eine Forms-Migration planen.

Ausgangslage, Voraussetzungen und Schlüssel-Anforderungen

Die vorangegangenen Fragen stammen aus einem Migrationsprojekt, bei dem die Triestram & Partner GmbH (**t&p**) ihr Forms basiertes Standard-Software-Produkt lisa.lims auf eine Java Open Source Architektur umgestellt hat. Bei lisa.lims handelt es sich um ein Labor-Informations- und Management-system, das bis zur Version 9 auf den „klassischen“ Oracle-Technologien basierte, nämlich der Datenbank sowie Forms und Reports. Das System ist nach den Architektur-Empfehlungen von Oracle gestaltet worden - das heißt: Die Domänen bzw. Business-Logik steckt in Form von PL/SQL-Logik in der Datenbank. Dies zeigt sich in folgendem Mengengerüst:

- 150 Forms
- 250 Reports
- 400 PL/SQL-Datenbank-Packages

Für die Version 10 wurde lisa.lims nach Java migriert. Dabei wurde die Oracle Datenbank beibehalten und Oracle Forms und Oracle Reports wurden durch Java Open Source Komponenten ersetzt – nämlich

- die Eclipse Rich Client Platform (RCP) bzw. die Eclipse Rich Ajax Platform (RAP) für die Benutzer-Oberfläche und
- die Eclipse Business Intelligence and Reporting Tools (BIRT) für die Berichte

Investitionsschutz

Bei der Auswahl der Migrations-Strategie spielte der große Umfang der vorhandenen PL/SQL-Logik in der Datenbank eine entscheidende Rolle, denn sie stellt eine beachtliche Investition von ca. 80 – 150 Personen-Jahren dar. Um diese Investition zu schützen, war es ausgeschlossen, die Business-Logik einer Big-Bang-Migration zu unterziehen, und sie „auf der grünen Wiese“ in Java auf der Mittel-Schicht neu zu implementieren. Das wäre wirtschaftlich und zeitlich nicht sinnvoll gewesen. Daher hatte die Wiederverwendung des vorhandenen PL/SQL eine zentrale Bedeutung in der Liste der Anforderungen an die migrierte Anwendung.

Die folgenden Abschnitte zeigen, welche technische Rolle die vorhandene PL/SQL-Logik in der alten und der migrierten Anwendung spielt.

Mehrbenutzer-Fähigkeit und PL/SQL

lisa.lims ist ein Mehrbenutzer-System, in dem jeder einzelne Nutzer dediziert an der Datenbank angemeldet wird:

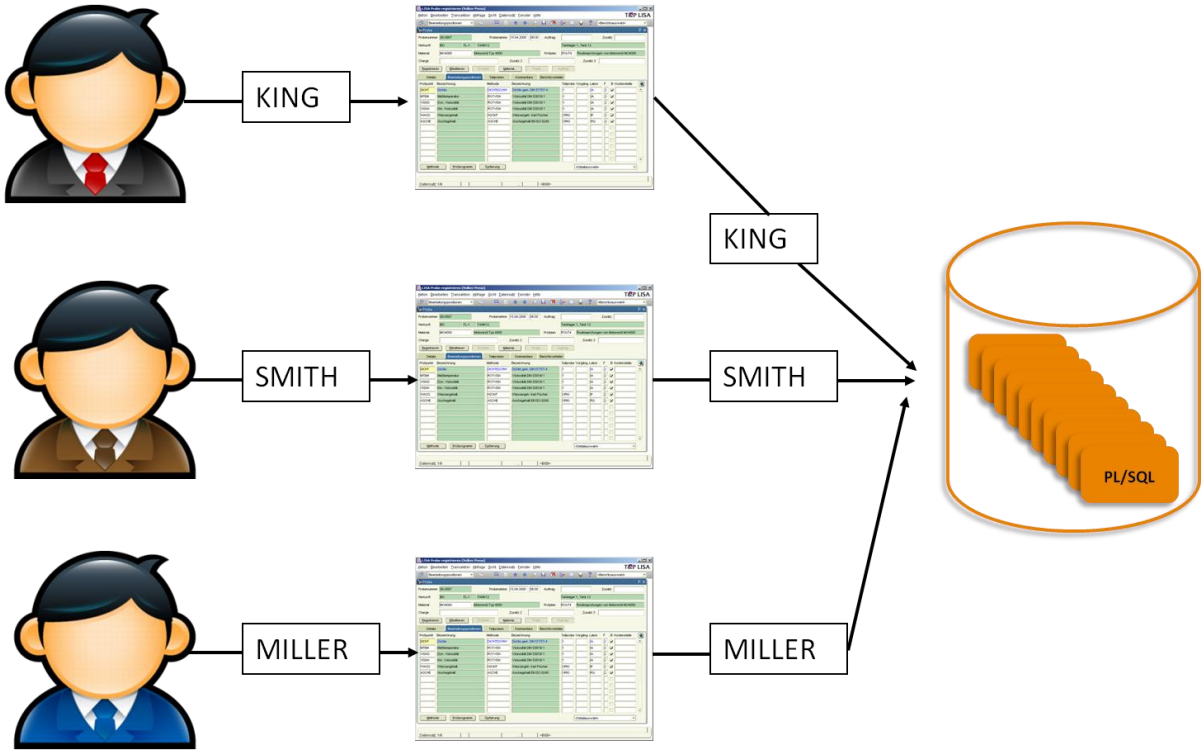

Abbildung 1: PL/SQL und Mehrbenutzerfähigkeit

Die PL/SQL-Logik in der Datenbank verwendet den Session-Kontext des angemeldeten Nutzers, unter anderem um mittels Package-Variablen die Datenbankänderungen nachzuverfolgen, die der Nutzer durchgeführt hat. Um diesen Mechanismus weiterhin nutzen zu können, muss das migrierte System ebenfalls in der Lage sein, jedem angemeldeten Benutzer eine individuelle Datenbank-Verbindung zuzuweisen.

Anbindung von Sub-Systemen an die Datenbank – PL/SQL als Tabellen-API

Die vorhandene PL/SQL-Logik dient unter anderem als Schnittstelle, über die Labor-Geräte schreibend auf die Datenbank zugreifen und so dort ihre Messdaten speichern. Damit dient das vorhandene PL/SQL in der Datenbank auch als API für den schreibenden Zugriff auf die Tabellen: Es stellt die Konsistenz von Schreibzugriffen auf die Datenbank sicher, auch von solchen, die außerhalb der Applikation erfolgen (z.B. externe Geräte mit Datenbank-Schnittstelle):

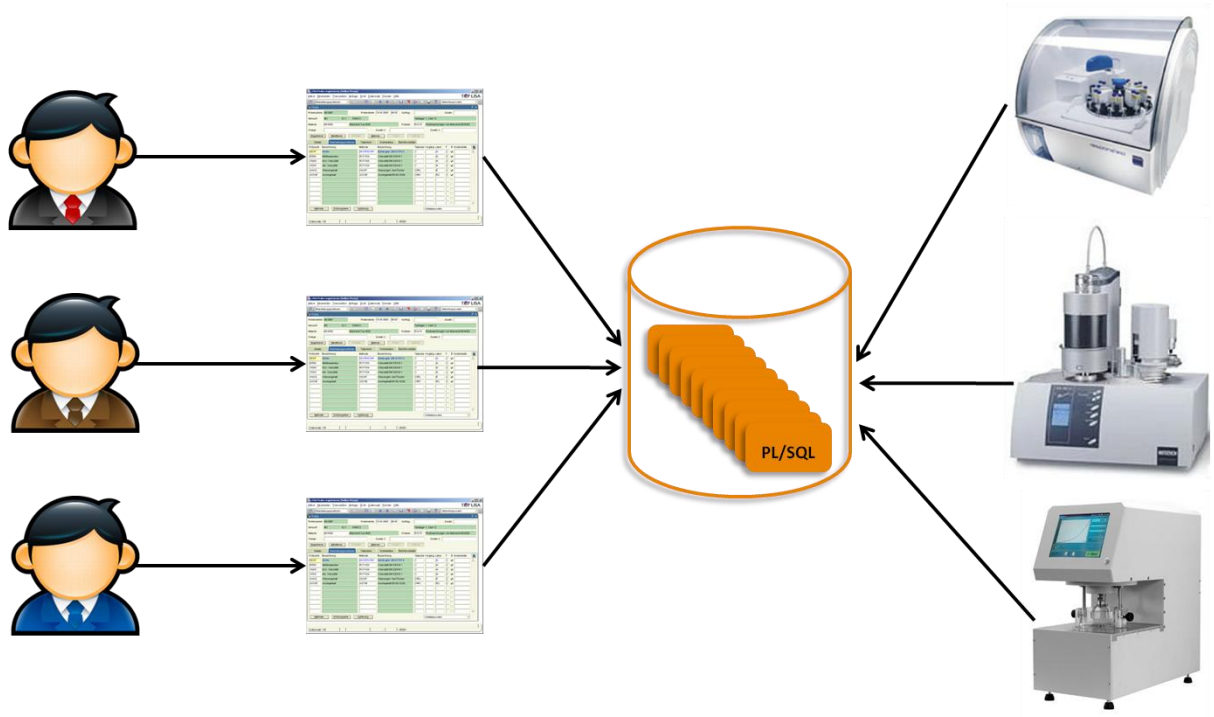

Abbildung 2: PL/SQL als Schnittstelle für Sub-Systeme

Erfolgen zusätzlich schreibende Zugriffe direkt auf die Tabellen der Datenbank, also etwa vom Application Server aus, ist unbedingt sicherzustellen, dass dabei die betroffenen Datensätze für die Dauer der Transaktion gesperrt werden (pessimistic locking), damit die Konsistenz der Daten insgesamt gewahrt bleibt.

Technische Anforderungen an die migrierte Anwendung in Bezug auf PL/SQL

Aus den geschilderten Rahmenbedingungen ergeben sich folgende technische Anforderungen an die migrierte Anwendung in Bezug auf das vorhandene PL/SQL in der Datenbank. Die migrierte Anwendung muss:

1. in der Lage sein, in der Datenbank gespeichertes PL/SQL aufzurufen.
2. jedem Applikationsbenutzer eine exklusive Datenbank-Verbindung zuteilen, die er über die Dauer der Anwendungssitzung beibehält.
3. den Benutzer unter seinem spezifischen Account in der Datenbank anmelden.
4. pessimistisches, exklusives Sperren von Datensätzen ermöglichen, wenn schreibende Zugriffe auf die Daten erfolgen.

Schwierigkeiten bei der Einbindung von PL/SQL in einer Java Enterprise Architektur (JEE)

In einer Vorstudie wurde unter anderem zunächst die Java Enterprise Technologie evaluiert. Es stellte sich heraus, dass die Umsetzung der oben genannten Anforderungen mittels der Java Enterprise Architektur aus folgenden Gründen nicht möglich war:

1. Inkompatibilität der Schichten-Trennung
   Der JEE Ansatz sieht vor, dass die Geschäftslogik grundsätzlich auf der Mittelschicht im Application Server implementiert wird – und zwar im EJB Container mittels Sessions Beans bzw. Entity Beans / Persistent Beans, deren Geschäftsdaten mittels Container-Managed Transactions in die Datenbank geschrieben werden. Dieser mittelschicht-zentrierte Architektur-Ansatz steht im grundsätzlichen Gegensatz zum datenbankzentrierten Ansatz, bei dem Geschäftslogik ganz oder teilweise in der Datenbank-Schicht implementiert ist. Dennoch wurde die Machbarkeit der Migration unter Verwendung der JEE Plattform geprüft, nicht zuletzt deshalb, weil JEE bereits weit verbreitetet ist und es sich um einen etablierten Standard handelt. Weitere Probleme führten jedoch letztendlich zu der Entscheidung, JEE für die Migration unserer Software nicht zu verwenden.

2. Container-Managed Persistence
   JEE sieht für die Geschäftslogik in Session Beans standardmäßig vor, dass Datenbank-Transaktionen durch den EJB-Container verwaltet werden. Das bedeutet, dass die Anwendung keine Kontrolle darüber hat, wann genau Datenänderungen in die Datenbank geschrieben werden. Damit ist ausgeschlossen, dass schreibende Zugriffe auf die Datenbank von außerhalb des Applikations-Servers erfolgen, also etwa durch PL/SQL in der Datenbank selbst. Das würde die Datenkonsistenz gefährden.
   Eine Alternative zur Container-Managed Persistence bieten die sogenannten User-Transactions. Hier kontrolliert die Anwendung bzw. der Anwender die Transaktion explizit – und zwar so, dass bei schreibenden Operationen an Daten die entsprechenden Datensätze in der Datenbank auch gesperrt werden. Dies würde ermöglichen, dass Schreibzugriffe auf die Datenbank vom Application Server aus und von den PL/SQL-Modulen in der Datenbank erfolgen, ohne dass dabei die Gefahr von Dateninkonsistenzen besteht. Bei den meisten Applikations-Servern war es zum Zeitpunkt der Technologiestudie möglich, die automatische Transaktionsverwaltung des Servers abzuschalten und User-Transactions zu verwenden. Jedoch war diese Konfiguration meist herstellerspe-

zifisch, was die Portierbarkeit der Anwendung stark einschränkt, und somit einen wesentlichen Vorteil von Java EE verspielt.

3. Connection Pooling
   In der JEE-Welt basiert die Verbindung zur Datenbank standardmäßig auf dem Prinzip des Connection Pooling: Der Applikations-Server startet eine definierte Anzahl von Verbindungen zur Datenbank, die ständig offen gehalten werden, um den Overhead des Verbindungsaufbaus zu minimieren. Führt der Benutzer in der Anwendung eine Aktion in der Datenbank aus, so nutzt der Applikations-Server dazu eine der Verbindungen aus dem Pool und gibt die Verbindung am Ende der Aktion wieder in den Pool zurück, wo sie dem nächsten Applikations-Benutzer zu Verfügung steht. Dieses Konzept hat zwei Folgen:

   a) Alle Benutzer der Applikation nutzen ein- und denselben Datenbank-Benutzernamen. PL/SQL-Logik in der Datenbank, die den Session-Kontext des Datenbankbenutzers verwendet oder Änderungsverfolgung betreibt, ist damit nicht mehr verwendbar.

   b) Um die vorhandene PL/SQL-Logik in der Datenbank wiederverwenden zu können, kommen für lisa.lims nur solche Applikations-Server in Frage, die in der Lage sind, eine Datenbank-Funktion in der gerade aktiven Datenbank-Session des Benutzers auszuführen. Jedoch nicht alle evaluierten Server waren dazu in der Lage, was von der Implementierung der Java Persistence API (JPA) des jeweiligen Servers abhängt. Ein Austausch der JPA-Implementierung ist zwar oft möglich, wird jedoch dann nicht mehr durch den Hersteller-Support abgedeckt.

4. Herstellerspezifische Erweiterungen des Applikations-Servers
   Die Spezifikation der Java Enterprise-Architektur bezieht sich auch auf den Applikations-Server. Sun/Oracle hat mit GlassFish eine Referenz-Implementierung vorgelegt. Andere Hersteller von Applikations-Servern weichen von diesem Standard ab, teilweise dadurch, dass sie spezifische Erweiterungen der Funktionalität bieten. Für die migrierte Anwendung relevante Spezifika fanden sich beim JBoss Application Server zum Beispiel im Bereich der Datensatzsperren. Da die migrierte Applikation jedoch auf möglichst vielen Applikations-Servern lauffähig sein soll, ist die Nutzung solcher Hersteller-Spezifika strikt zu vermeiden.

Am Ende der Technologie-Studie zeigte sich deutlich: Eine JEE-konforme Architektur unter Verwendung eines JEE-konformen Applikations-Servers stand einer der Schlüssel-Anforderungen bei der Migration im Wege, nämlich der Wiederverwendung der vorhandenen PL/SQL-Logik in der Datenbank der migrierten Anwendung.

Genau diese gesuchte Möglichkeit bietet das Spring-Framework in Kombination mit dem Objekt-relationalen Mapper EclipseLink. Diese Lösung wird in den folgenden Abschnitten näher erläutert.

Integration von PL/SQL mittels Spring und EclipseLink

Aufgrund der beschriebenen Ergebnisse der Technologie-Studie hat sich **t&p** für folgenden Ansatz entschieden:

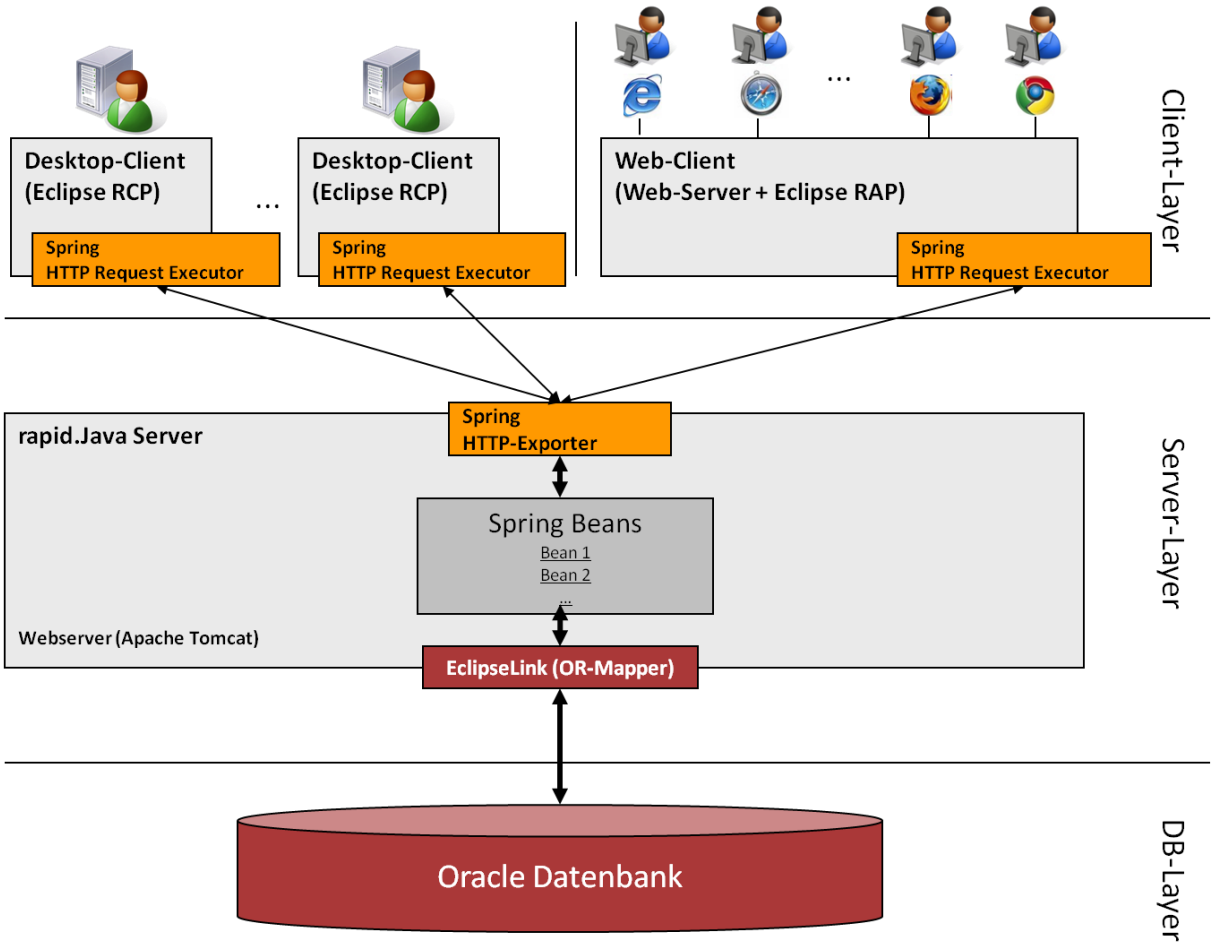

Abbildung 3: Remoting mit Hilfe des Spring Frameworks

Im gewählten Ansatz implementieren die Spring-Beans auf der Mittelschicht die Zugriffe auf die Datenbank. Das Dispatcher-Servlet des Spring Frameworks bindet die einzelnen Spring-Beans dediziert an die einzelnen HTTP-User-Sessions, die in einem servlet-fähigen Webserver laufen. Der jeweilige Client greift auf seine Web-Session mittels des Spring-HTTP-Invokers zu:

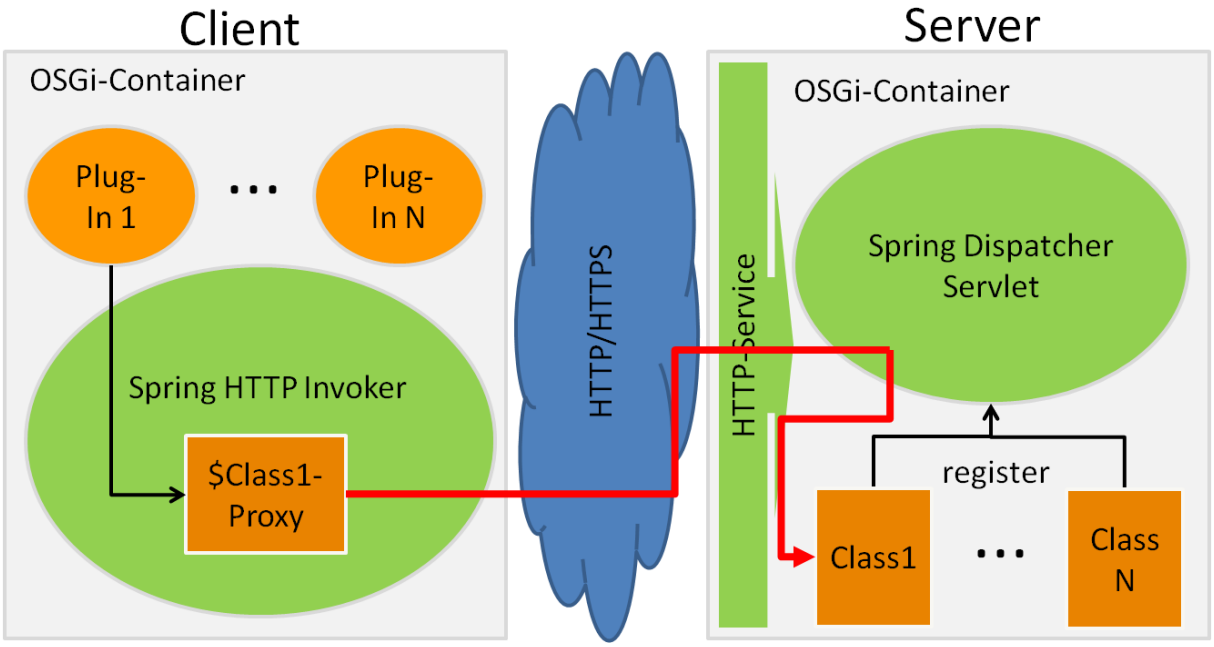

Abbildung 4: Remote-Zugriff auf Logik im Application Server mittels Spring

Verwendung des Objekt-Relationalen Mappers EclipseLink

EclipseLink (z. Zt. in der Version 2.2 in Verwendung) ermöglicht es,

- Datensätze in der Datenbank pessimistisch zu sperren
- PL/SQL in der Datenbank aufzurufen
- jedem Applikationsbenutzer eine exklusive Datenbank-Verbindung zuzuteilen und den Benutzer unter seinem spezifischen Account in der Datenbank anzumelden.

Damit erfüllt EclipseLink alle technischen Anforderungen, die erforderlich sind, um das vorhandene PL/SQL in der neuen Java-Anwendung wiederzuverwenden.

Darüber hinaus implementiert EclipseLink die JPA Spezifikation, sodass auch viele nützliche JPA-Features genutzt werden können. Für die JPA Spezifikation 2.0 ist EclipseLink die Referenz-Implementierung. Da EclipseLink auf dem OR-Mapper „Toplink“ von Oracle basiert, ist es sogar möglich, einige herstellerspezifische Funktionen der Oracle Datenbank zu nutzen. Damit war EclipseLink unsere erste Wahl.

Fazit

Welche Vorteile bietet der gewählte Ansatz?

1. Der Ansatz verleiht der migrierten Java-Anwendung die datenbank-relevanten Eigenschaften, die in der Welt von Oracle-Forms-Anwendungen etablierter und geschätzter Standard sind: Die Mehrbenutzer-Fähigkeit inklusive des pessimistischen DML-Sperrverhaltens und der individualisierten

Anmeldung an der Datenbank sowie die Integration von vorhandener PL/SQL-Logik. Diese Features sind für eine datenbank- bzw. OLTP-Anwendung grundlegend.

2. Die Kombination von EclipseLink mit dem Remoting des Spring Frameworks ermöglicht es, einen Applikations-Server zu implementieren, der die zuvor genannten Datenbank-Anforderungen erfüllt und dabei ohne JEE-Overhead auskommt. Diese Lösung zeichnet sich dadurch aus, dass sie leichtgewichtig und äußert flexibel ist: Der Server nutzt OSGi als Laufzeit-Umgebung und kann damit sowohl „standalone“ laufen, als auch in jeden servlet-fähigen Web- bzw. Applikationsserver als Standard-Webanwendung (WAR) deployed werden. Somit ist die gewählte Architektur unabhängig von den Spezifika einzelner Applikations-Server, es wird lediglich ein Servlet-Container benötigt.

3. Die Integration von vorhandener PL/SQL-Datenbank-Logik bedeutet einen Schutz der bereits getätigten Investition und einen geringeren zeitlichen und finanziellen Aufwand bei der Migration.

4. Der Ansatz ermöglicht eine "smart migration“ – und zwar im Sinne eines schrittweisen, fließenden Übergangs von der alten auf die neue Technologie-Plattform. Die Grundlage hierfür ist die Wiederverwendung der vorhandenen PL/SQL-Business-Logik: Einzelne besonders zentrale Dialoge können bereits auf die neue Technologie für die Benutzer-Oberfläche umgestellt werden, während andere Dialoge zunächst weiter auf Oracle Forms basieren. So können Alt und Neu koexistieren, bis die gesamte Migration abgeschlossen ist. Anschließend kann die Neu-Implementierung der Business-Logik in Java ins Auge gefasst werden.

Somit verbindet der vorgestellte Ansatz technisch das Beste aus zwei Welten – nämlich die Beibehaltung der transaktions-orientierten Features auf Seiten der Oracle Datenbank, mit einer gleichzeitig plattformunabhängigen, leichtgewichtigen und flexiblen Java-Architektur.

Kontaktadresse:

Thomas Haskes & Jens Hüttemann

Triestram & Partner GmbH (**t&p**)

Kohlenstraße 55

D-44795 Bochum

Telefon: +49 (0) 234-9 43 75 - 0

Fax: +49 (0) 234-45 22 06

E-Mail t.haskes@t-p.com bzw. j.huettemann@t-p.com

Internet: www.t-p.com