Migration der Datenbankzugriffsschnittstelle in Client/Server-Anwendungen

Christian Böhmer
iSYS Software GmbH
München

Björn Grimm
Competence Center Wirtschaftsinformatik
Hochschule München

Schlüsselworte:

JEE, EJB, JPA, JDBC, Hibernate, Oracle Database, Anwendungsarchitektur, Technologiebewertung, Migration

Einleitung und Motivation

Mit zunehmender Komplexität werden Anwendungssysteme heute üblicherweise in mehrere Softwareschichten strukturiert. Hierbei werden den einzelnen Schichten unterschiedliche Bedeutungen zugerechnet. Die meisten Projektvorgaben enthalten zwar genaue Einzelheiten über die technische Realisierung der Oberfläche, die zu verwendenden Zugriffsprotokolle oder die einzusetzende Servertechnologie, die wenigsten jedoch messen der Datenbankzugriffsschnittstelle größere Bedeutung bei. Dabei kann sie leicht mehr als 20% des gesamten Programmcodes ausmachen und ist somit sicher eine der wichtigsten Architekturschichten einer betrieblichen Anwendung.

Um die Zukunftsfähigkeit einer Anwendung zu gewährleisten, bedarf es einer stetigen Untersuchung und gegebenenfalls Anpassung der eingesetzten Technologien. Neue Techniken beinhalten teilweise erhebliche Vorteile, so dass eine Migration hin zu einer neuen Technologie im Rahmen der Architekturentwicklung von Zeit zu Zeit in Betracht gezogen werden sollte. Durch seine zentrale Bedeutung kann ein Austausch der Datenbankzugriffsschicht zu einer Verbesserung der Anpass- und Wartbarkeit, sowie der Leistung und auch zu einer geringeren Fehleranfälligkeit durch Verwendung von Standardprodukten führen. Manche Technologien haben auch ganz spezifische Vorteile wie beispielsweise eine besonders gute Integration in die Testautomatisierung. Neben diesen positiven Aspekten sollten jedoch auch die Aspekte Risiko und Realisierbarkeit berücksichtigt werden. Erst nach einer Migrationsanalyse kann eine Empfehlung ausgesprochen werden, ob ein Wechsel zu einer neuen Technologie technisch als sinnvoll erachtet werden kann. Diese Analyse erweist sich aber bei größeren Anwendungen als aufwändig.

Dieser Beitrag befasst sich mit der Migration von Datenbankzugriffsschnittstellen, wobei Java-Technologien im Fokus stehen. Nach einer kurzen Einführung in Zugriffstechniken wird für ein konkretes Projekt eine Analyse für eine Migration einer klassischen Zugriffsschicht hin zu einer JPA-basierte Zugriffsschicht vorgestellt.

Datenbankzugriff in der Architekturbetrachtung

Für den Zugriff auf eine Datenbank existieren bereits eine Vielzahl unterschiedlicher Techniken, denen jedoch in der Java-Welt eine gemeinsame Basis zugrunde liegt. Sie alle verwenden JDBC zur Kommunikation mit der Datenbank.

Klassische Zugriffstechniken

Als klassische Zugriffstechniken bezeichnen wir Techniken, in denen nur durch Design Patterns gestützt weitestgehend direkt JDBC oder bei anderen Sprachumgebungen eine entsprechende API verwendet wird. In einigen Fällen wird der Zugriff auf JDBC durch Hilfsklassen gekapselt, jedoch erfolgt die Definition eines Datenbankaufrufs generell über SQL-Anweisungen.

Im Folgenden werden die wichtigsten, klassischen Zugriffstechniken kurz vorgestellt:

- Mit dem *Data Access Object (DAO)* Pattern erfolgt der Zugriff auf die Datenbank über zentrale Zugriffsklassen. Daher wird dieses Entwurfsmuster auch oft als *Data Mapper* [1] bezeichnet. Die Zugriffsklassen können hierbei Datenbankeinträge laden oder aber übergebene Objekte speichern und ändern.
- Beim *Active Record* Pattern [1] repräsentiert ein Objekt einen Datensatz. Die Instanz eines Objektes kann somit exakt einer Tabellenzeile zugeordnet werden. Für das Laden und Speichern einzelner Objekte verfügen die Zugriffsklassen über *find*- bzw. *save*-Methoden, so dass sich die gesamte Zugriffslogik in den Objekten selbst befindet.
- Nahezu identisch zum Active Record Pattern ist das *Row Data Gateway* [1]. Auch hier erfolgt der Zugriff auf die Datenbank über konkrete Objektinstanzen als Repräsentanten eines Datensatzes. Im Gegensatz zum Active Record Pattern beinhaltet das Row Data Gateway jedoch keinerlei Geschäftslogik.
- Das *Table Data Gateway* [1] verwaltet mit einem Objekt ganze Tabellen einer Datenbank. Eine Instanz einer solchen Implementierung enthält somit alle Datensätze dieser Tabelle.

Als Sonderform sei hier noch *Spring JDBC Template* [2] erwähnt. Hier erfolgt der Zugriff nicht unmittelbar direkt über JDBC, sondern gekapselt über spezielle Hilfsklassen. Mit diesen Hilfsklassen lassen sich auch Objektinstanzen aus Rückgabewerten erzeugen, so dass die Implementierung auf ein Minimum reduziert wird.

ORM-Ansatz

Beim ORM-Ansatz - ORM steht für Object-Relational Mapping - werden die Objekte direkt auf die Datenbank abgebildet. Das Abbilden dieser Objekte kann hierbei entweder deklarativ über Konfigurationsdateien oder aber direkt im Programmcode erfolgen. Im Gegensatz zum Active Record Pattern muss beim ORM-Ansatz keine Zugriffslogik implementiert werden, da dies vom ORM-Framework zur Laufzeit erledigt wird. Für das Ausführen von Datenbankabfragen kann neben SQL eine native Abfragesprache verwendet werden. Diese ist meist sehr ähnlich zu SQL und hat den Vorteil, unabhängig vom zugrundeliegenden Datenbanksystem zu sein. Ein Austausch des Datenbanksystems könnte somit ohne Anpassung des Programmcodes erfolgen.

Derzeit existieren die beiden Defacto-Standards *Java Persistence API (JPA)* und *Java Data Object (JDO)*, welche durch Sun Microsystems im Java Community Process durch JSR 220 (JPA) und JSR 243 (JDO) spezifiziert sind. Neben Hibernate implementieren unter anderem auch TopLink und EclipseLink den JPA Standard. Auch für JDO existieren einige Implementierungen wie beispielsweise

Apache JDO oder DataNucleus. Da beide Standards im Grunde den gleichen ORM-Ansatz verfolgen, wird an dieser Stelle auf einen detaillierten Vergleich verzichtet.

Die betrachtete Anwendung

Das hier betrachtete Anwendungssystem wurde bereits in einem DOAG Vortrag [3] im Vorjahr näher vorgestellt. Es handelt sich hierbei um eine Buchhaltungssoftware für Immobilienverwalter, Hausbesitzer und Wohnungseigentumsgemeinschaften mit einer integrierten Anbindung an den Zahlungsverkehr. Neben der Erfassung und Verwaltung von Wirtschaftseinheiten mit Wohnungen, einschließlich der Verwaltung aller Nebenbücher für die Immobilienverwaltung, dient das System auch der Erstellung von Jahres- oder Nebenkostenabrechnungen. Durch die integrierte Anbindung an die kontoführenden Bankensysteme ist eine automatisierte Übertragung von Kontoinformationen und Inkassoaufträgen möglich. Das System stellt für mehr als 1000 Benutzer eine Online-Schnittstelle bereit und wickelt die länger andauernden Berechnungen (Jahresabrechnungen, tägliche Inkasso) im nächtlichen Batchbetrieb ab.

Da das Gesamtsystem aus mehreren Servern besteht, wird im Folgenden nur der für die Datenbankzugriffsschnittstelle relevante Server betrachtet.

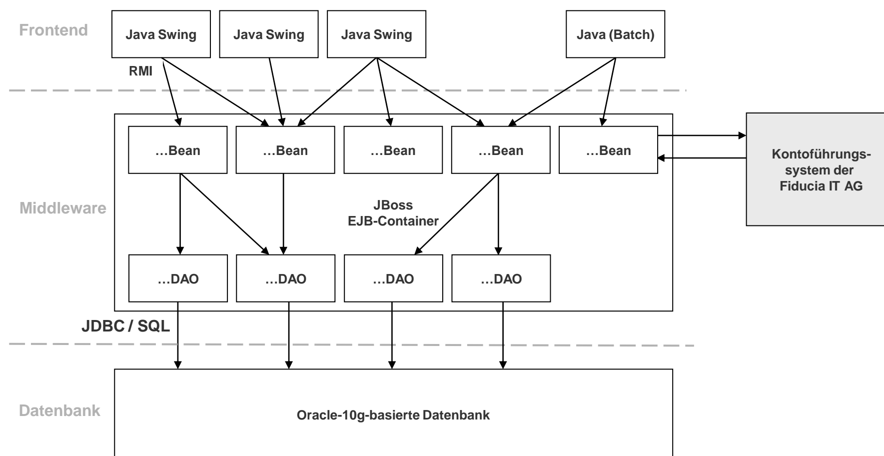

Abb. 1: Vereinfachte Architektur des Anwendungssystems [3]

Die Abbildung 1 zeigt die vereinfachte Architektur des relevanten Anwendungssystems. Hierbei interagieren mehrere Java Swing Clients mit einem JBoss JEE Server. Die einzelnen Client-Anfragen werden im Server durch Stateless Session Beans verarbeitet. Für den Zugriff auf die Datenbank werden leicht modifizierte Data Access Objects verwendet, so dass der Datenbankzugriff in einer eigenen Architekturschicht stattfindet. Als Datenbanksystem kommt Oracle 10g zum Einsatz.

Das Anwendungssystem wird seit 2002 entwickelt und umfasst derzeit insgesamt etwa 560.000 Zeilen Programmcode verteilt auf 2095 Klassen. Die Datenbank enthält 263 Tabellen und hat ein Datenvolumen von mehr als 60 GB.

Migrationsanalyse

Der erste Schritt bei der Durchführung einer Migrationsanalyse ist eine detaillierte Zieldefinition. Hierin werden nicht nur die grundlegenden Bedingungen festgelegt, sondern auch basierend auf einer Marktanalyse die zu verwendenden Produkte bestimmt. Im konkreten Fall wurde festgelegt, dass der Ressourcenverbrauch und die Verarbeitungszeit bei einer Umstellung des Datenbankzugriffs vor allem bei intensiv genutzten Programmteilen nicht mehr als 20% steigen darf. Als ORM-Produkt entschied man sich für Hibernate als eine Implementierung von JPA, da JPA bereits Bestandteil der EJB Spezifikation 3.0 ist und die Anwendung ebenfalls in EJB realisiert wurde.

Die Migrationsanalyse offenbarte einige Probleme, die bei einer Migration zu JPA entstehen würden. Die wichtigsten werden im Folgenden kurz vorgestellt und mögliche Lösungsansätze werden erörtert.

Als erstes Problem stellte sich die Verwendung von primitiven Datentypen bei dem Domänenmodell heraus. Das Domänenmodell bildet bei JPA ein Abbild der Datenbank und ist somit dafür verantwortlich, die Daten aus der Datenbank an den Client bzw. allgemein betrachtet an den anfragenden Prozess zu transportieren. Aus diesem Grund werden die Klassen des Domänenmodells auch häufig als Data Transfer Objects (DTO) bezeichnet. Bei der Verwendung von primitiven Datentypen wie beispielsweise "int" oder "float" kann Hibernate keine NULL-Werte zuweisen, auch wenn dies durch die Datenbank bereits gar nicht möglich wäre. Bei einer exzessiven Verwendung von primitiven Datentypen kann dies durchaus ein großes Problem werden, da sowohl die Businesslogik als auch der Client entsprechend angepasst werden müssten. Alternativ könnte das Problem dadurch behoben werden, dass das Abbilden der Attribute auf die Datenbankspalten nicht per Annotation an der Attributdeklaration erfolgt, sondern über entsprechende Annotationen an den Getter- und Setter-Methoden des Attributs. Dies reduziert jedoch den Vorteil der besseren Lesbarkeit des Programmcodes nicht unerheblich.

Ein weiteres Problem, dass auf gleiche Weise gelöst werden kann, ergibt sich, wenn in den Getter- oder Setter-Methoden des Domänenobjekts weitere Logik implementiert ist. Als Beispiel sei hier der Fall genannt, dass die Getter-Methode eines Datumsfeldes maximal das heutige Tagesdatum zurückliefert. Solche Methoden müssten entweder vorab bereinigt werden, so dass es sich vor der Migration um reine POJOs handelt, oder aber man schreibt zusätzliche Getter- und/oder Setter-Methoden mit entsprechender Annotation der Datenfeldzuweisung.

Bei sicherheitsrelevanten Anwendungen stellt sich die Frage des Risikos, wenn eine detaillierte Datenbankbeschreibung in Form von Annotation im Domänenmodell zu einem eventuell nicht geschützten Client übertragen wird. Ein Angreifer könnte das auch im Client zur Verfügung stehende Domänenmodell per Decompiler wieder in lesbaren Programmcode übertragen und anschließend die gewonnenen Erkenntnisse für einen Angriff nutzen. In sicherheitsrelevanten Anwendungen sollte aus diesem Grund auf die Verwendung von Annotationen für die Datenfeldzuweisung verzichtet werden und statt dessen nur serverseitig vorhandene Konfigurationsdateien verwendet werden.

Ein exemplarisch durchgeführter Leistungsvergleich des bisher eingesetzten DAO-Patterns mit Hibernate zeigte die Unterschiede beider Techniken unter identischen Voraussetzungen. Hierbei sollten in einem einzigen Aufruf Daten aus insgesamt 15 Tabellen geladen und an den Client transportiert werden. Als Testfall wurde bewusst ein aufwändiger und leistungskritischer Prozess gewählt, um das Ausmaß der wirklichen Problemfälle besser einschätzen zu können. Die Messung umfasste 10 Einzelmessungen und erfolgte im Client unter Verwendung des stets gleichen Aufrufs. Alle programminternen Zwischenspeicher wurden deaktiviert und zwischen den Einzelmessungen

wurden mindestens 30 Sekunden andere Abfragen gestartet, um auch ein Zwischenspeichern der Abfrageergebnisse in der Datenbank zu minimieren.

Messung	1	2	3	4	5	6	7	8	9	10	Mittelwert
DAO-Pattern	93	91	82	77	101	76	68	64	89	70	81
Hibernate	427	328	369	375	353	453	430	313	349	354	375

Abb. 2: Leistungsbetrachtung DAO-Pattern und Hibernate [4], alle Angaben in [ms]

Die Messreihe in Abbildung 2 zeigt die benötigte Zeit der beiden Datenbankzugriffstechniken in Millisekunden. Hierbei ist leicht zu erkennen, dass Hibernate im Vergleich zum DAO-Pattern im Schnitt die über 4-fache Zeit benötigt, um alle Daten aus der Datenbank zu laden. Dieser Wert könnte sich durch Optimierung vielleicht etwas reduzieren lassen, in leistungskritischen Prozessen ist Hibernate jedoch dem DAO-Pattern stark unterlegen. Einzig die Integration des Preload-Patterns [5] könnte Hibernate zu teils massiv kürzeren Ladezeiten verhelfen. Dies würde jedoch eine weitreichendere Analyse und Anpassung vieler Anwendungsteile erfordern, bei dem auch die Schnittstellen zum Server erweitert werden müssten.

Im letzten Schritt innerhalb der Migrationsanalyse fand ein Vergleich des Programmcodes beider Technologien im Hinblick auf Übersicht und Wartbarkeit statt. Da bei der Verwendung von Hibernate die typischen Datenbankzugriffsklassen des DAO-Patterns auf Kosten einer Konfigurationsdatei oder einiger Annotationen entfallen, scheint auf den ersten Blick Hibernate klar im Vorteil zu sein. Wenn man jedoch die bereits aufgeführten Probleme inklusive der nötigen Problemlösungen berücksichtigt, dann fällt der Vorteil nicht mehr so eindeutig aus. Speziell das Preload-Pattern würde zwar das Laden von Daten ungemein flexibel gestalten, jedoch geschieht dies immer auf Kosten der Übersicht und Einfachheit.

Zusammenfassung und Empfehlung

Die Ergebnisse unserer Analyse haben gezeigt, dass eine Migration zu einer ORM-basierenden Datenbankzugriffsschicht unter bestimmten Voraussetzungen möglich ist. Die dabei auftretenden Probleme sind, mit Ausnahme der Leistungsunterschiede, lösbar. Grundvoraussetzung für eine Migration sollte jedoch in jedem Projekt sein, dass der Datenbankzugriff in einer separaten Schicht erfolgt. Interessanter Weise ließe sich auch das bereits im betrachteten Projekt verwendete DAO-Pattern mit Hibernate kombinieren, so dass der Datenbankzugriff über Hibernate-DAOs erfolgen könnte. Der Vorteil dieser Variante bestünde darin, dass sich die konkrete DAO-Implementierung sehr einfach durch andere Implementierungen wie beispielsweise Mock-Objekte austauschen ließe.

In welchen Fällen eine Migration zum ORM-Ansatz sinnvoll erscheint, muss für jedes Projekt differenziert betrachtet werden. Unsere Erfahrungen haben gezeigt, dass vor allem beim Zugriff auf komplexe Tabellenstrukturen und beim Laden von großen Datenmengen zum Teil sehr lange Zugriffszeiten unvermeidbar sind. Hierfür gibt es bereits auch einige Lösungsansätze, jedoch stellt sich bei einigen dieser Möglichkeiten die Frage, ob eine Migration tatsächlich gewinnbringend ist. Speziell in leistungskritischen Prozessen oder während einer Batch-Verarbeitung können durch eine Migration große Leistungsprobleme entstehen.

Kontaktadresse:

Christian Böhmer
iSYS Software GmbH
Lucile-Grahn-Str. 37
D-81675 München

Telefon:	+49 (0) 89-4623 2821
Fax:	+49 (0) 89-4623 2814
E-Mail	c.boehmer@isys-software.de
Internet:	www.isys-software.de

Björn Grimm
Hochschule München
Fakultät für Informatik und Mathematik
Lothstraße 34
D-80334 München

E-Mail	b.grimm@hm.edu
Internet:	www.hm.edu

Die Arbeit wurde unterstützt durch das Competence Center Wirtschaftsinformatik der Hochschule München.
Sprecher: Prof. Dr. Peter Mandl

Literaturverzeichnis:

[1] Fowler, M.: Patterns of Enterprise Application Architecture, Addison-Wesley Professional, 2002
[2] Spring Framework: http://static.springsource.org/spring/docs/2.0.x/reference/jdbc.html, letzter Zugriff am 01.09.2010
[3] Mandl, P.: Erfolgreicher Einsatz von JEE/EJB in einem Bankenprojekt, DOAG-Konferenz, 2009
[4] Grimm, B.: Analyse und Implementierung unterschiedlicher Object Relation Mapping Methoden am Beispiel einer Bankanwendung, Bachelorarbeit, 2010
[5] Kohl, J.: Hibernate Preload Pattern, Java Magazin, Ausgabe 4.2008