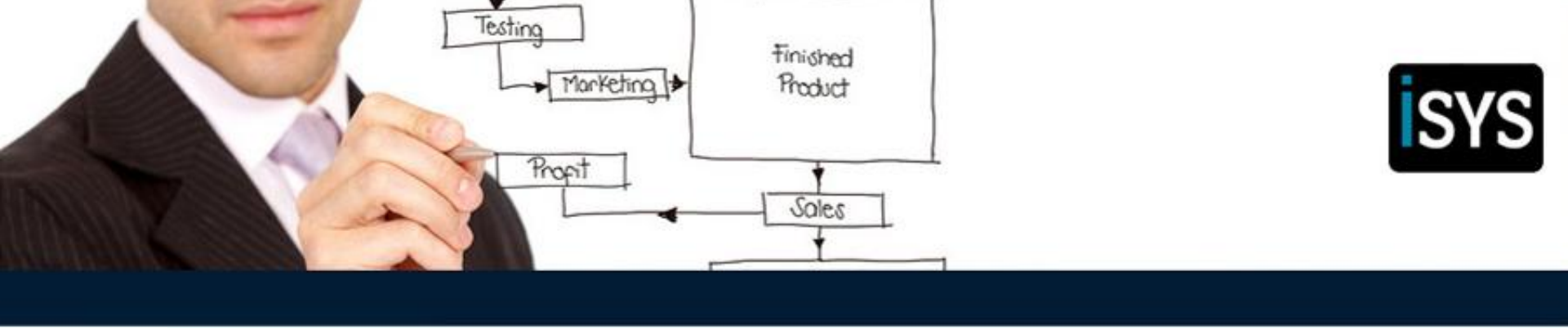


Migration der Datenbankzugriffsschnittstelle in Client-/Server-Systemen

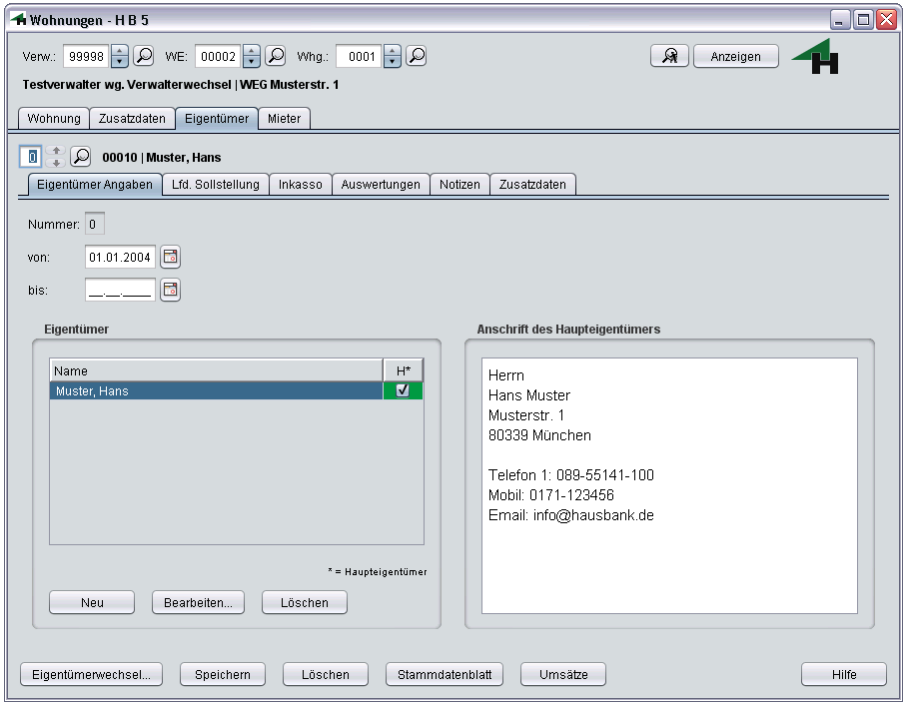
Christian Böhmer, iSYS Software GmbH
Björn Grimm, Hochschule München

Agenda

- **Einleitung**
- **Theorie – Zugriffstechniken**
  - Klassische Zugriffstechniken
  - ORM-Ansatz
- **Praxis**
  - Migrationsanalyse
  - Zielsetzung
  - Durchführung
  - Leistungsbetrachtung
  - Codevergleich
- **Fazit**

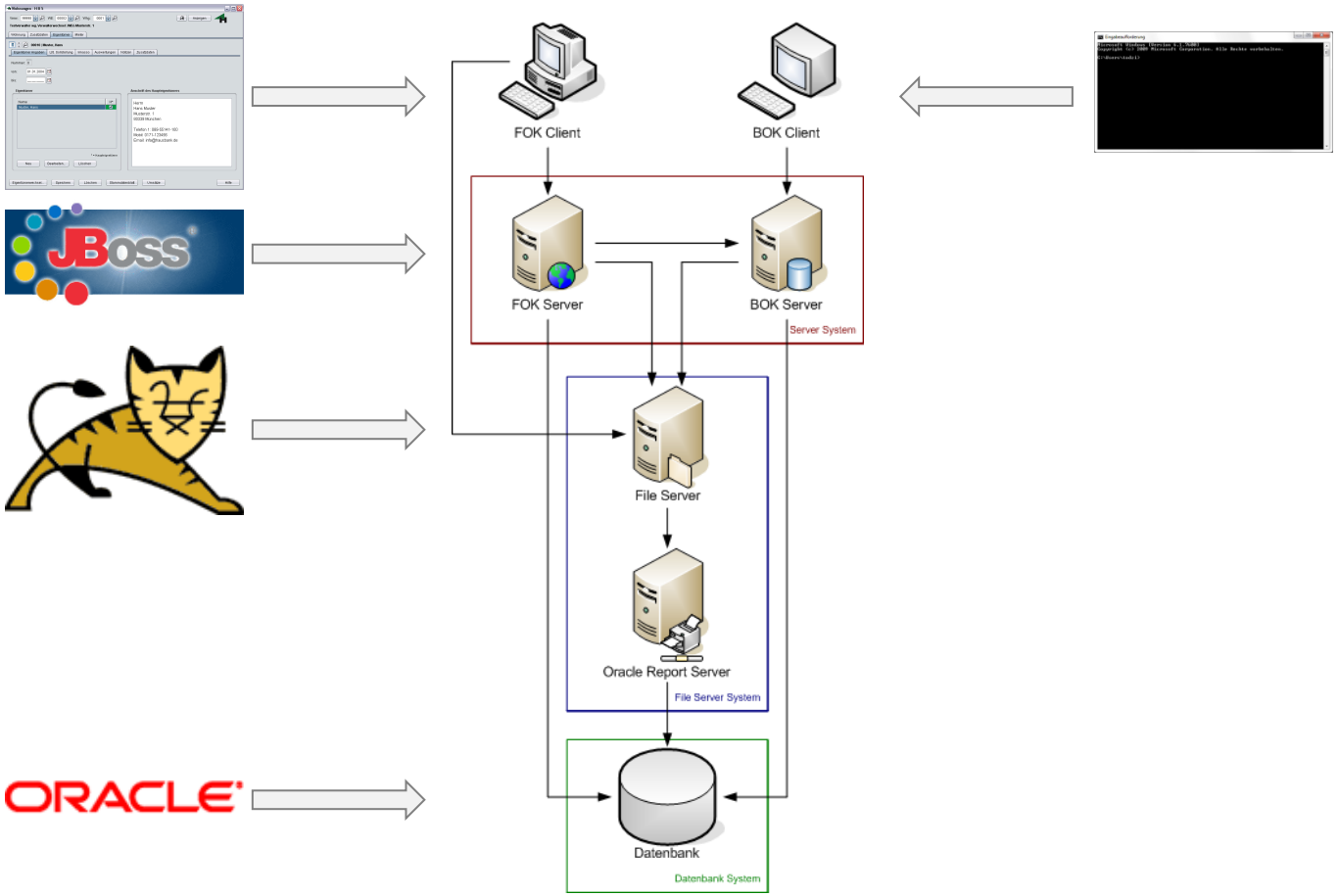
Einleitung

- **Projekt „Verwalter-Service“**
  - Auftraggeber: Hausbank München
  - Projektumfang: 8-15 Mitarbeiter, ~550.000 LOC (Java), ~250 Tabellen



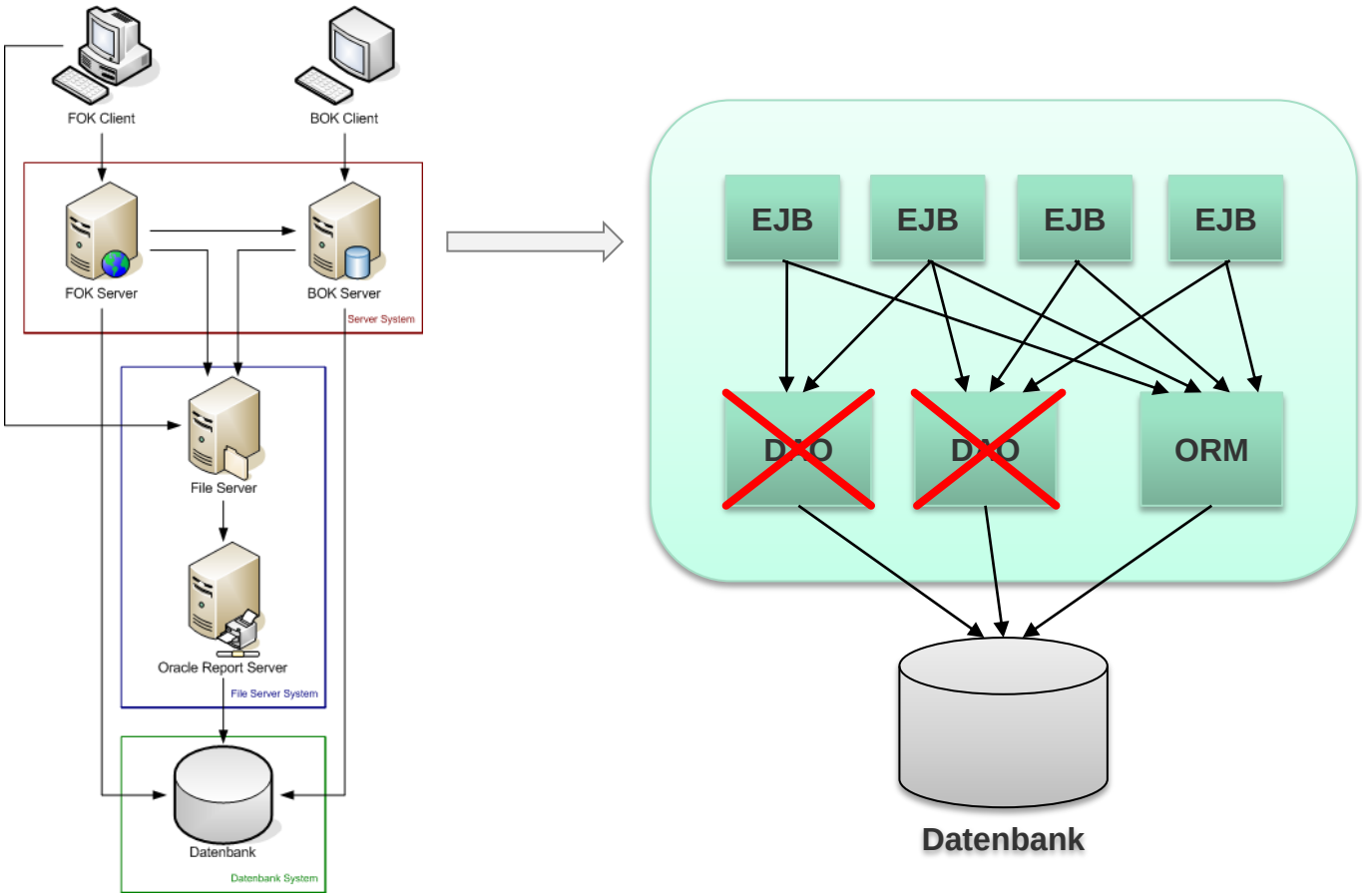
Einleitung

- **Systemarchitektur**



Einleitung

- **Aufgabenstellung**



Einleitung

- **Erhoffte Vorteile**
  - Bessere Anpass- und Wartbarkeit der Datenbankzugriffsschnittstelle
  - Verbesserte Leistung
  - Weniger Fehler

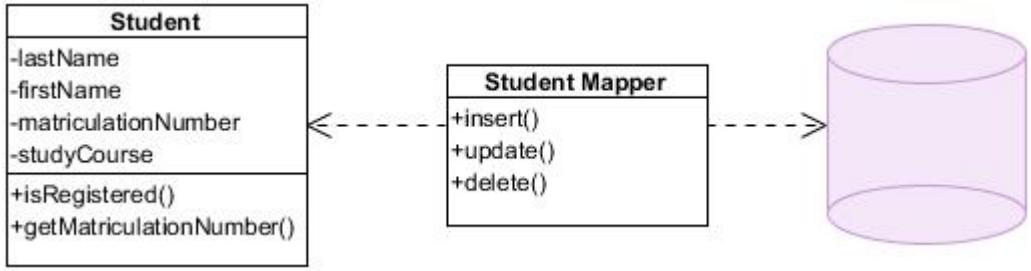
- **Befürchtete Nachteile**
  - Risiko
  - Realisierbarkeit
  - Teilweise wesentlich schlechtere Leistung

Theorie - Zugriffstechniken

- **Klassische Zugriffstechniken**
  - Weitestgehend direkte Verwendung von JDBC, gestützt durch Design Patterns
  - Kapsellung der JDBC Zugriffe in eigenen Hilfsklassen
  - Definition eines Datenbankaufrufes erfolgt generell durch SQL

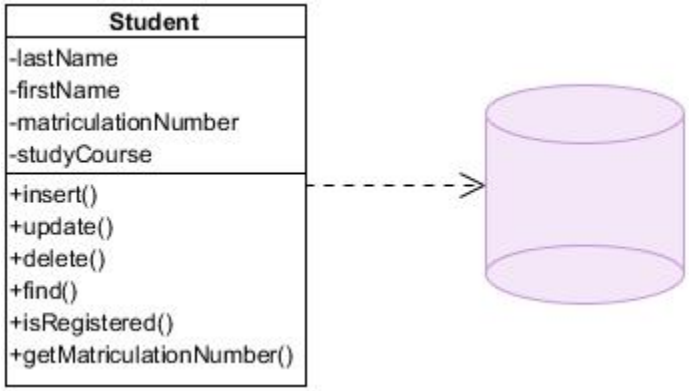
Theorie - Zugriffstechniken

- **Data Access Object (DAO) / Data Mapper**
  - Zugriff erfolgt über zentrale Zugriffsklassen
  - Zugriffsklassen können Datenbankeinträge
    - laden
    - übergebene Objekte speichern bzw. ändern



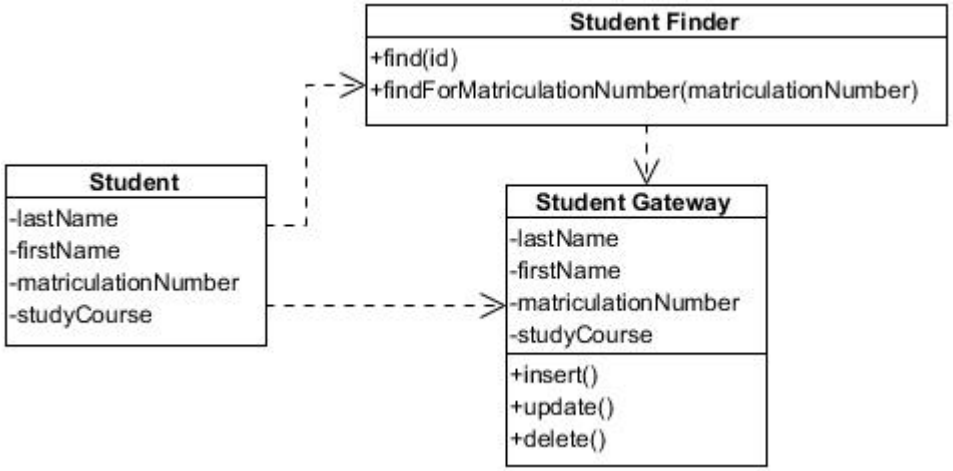
Theorie - Zugriffstechniken

- **Active Record Pattern**
  - Ein Objekt repräsentiert einen Datensatz
  - Gesamte Zugriffslogik ist im Objekt selbst enthalten
    - find Methode
    - save Methode
    - insert Methode



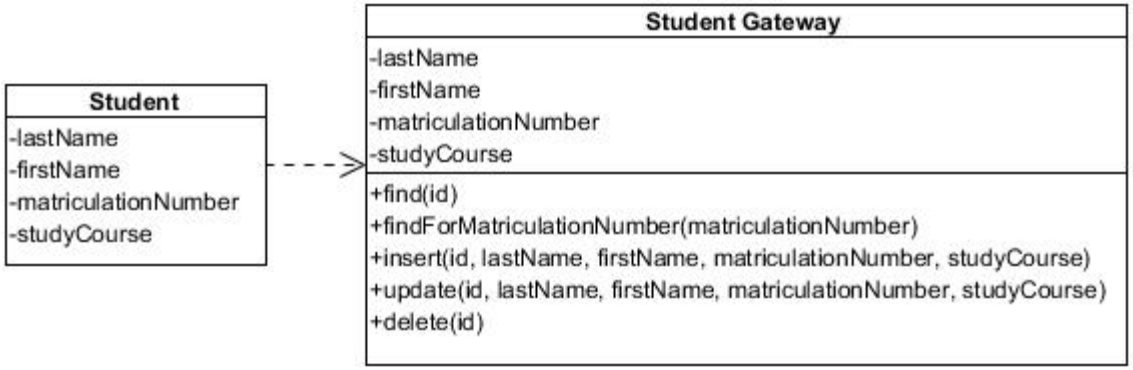
Theorie - Zugriffstechniken

- **Row Data Gateway**
  - Nahezu identisch zum Active Record Pattern
  - Enthält keine Geschäftslogik
  - Mehr objektorientierter Ansatz, da Kapsellung durch Interfaces erfolgt



Theorie - Zugriffstechniken

- **Table Data Gateway**
  - Verwaltet ganze Tabellen einer Datenbank in einem Objekt
  - Eine Instanz enthält somit alle Datensätze dieser Tabelle



Theorie - Zugriffstechniken

- **ORM-Ansatz (1)**
  - ORM ist die Abkürzung für Object-Relational-Mapping
  - Objekte werden direkt auf die Datenbank abgebildet
  - Implementierung der Zugriffslogik nicht notwendig
  - Wird zur Laufzeit vom ORM-Framework vorgenommen
  - Abbildungsbeschreibung kann erfolgen durch
    - deklarative Konfigurationsdateien
    - Annotationen im Programmcode

Theorie - Zugriffstechniken

- **ORM-Ansatz (2)**
  - Bietet den Vorteil, dass das zugrundeliegende Datenbanksystem ohne Anpassung am Programmcode ausgetauscht werden kann
  - Datenbankabfragen erfolgen über
    - SQL
    - native Abfragesprache, ähnlich zu SQL
  - Vertreter des ORM-Ansatzes
    - Java Persistence API (JPA)
    - Java Data Object (JDO)
    - Container Managed Persistence (CMP)

Theorie - Zugriffstechniken

- **Java Persistence API**
  - Spezifiziert im Java Community Process, JSR 220
  - Frameworks verwenden Reflection für die Realisierung
  - Bekannte Implementierungen
    - Hibernate
    - EclipseLink bzw. TopLink

Theorie - Zugriffstechniken

- **Java Data Object**
  - Spezifiziert im Java Community Process, JSR 243
  - Frameworks verwenden Byte Code Manipulation für die Realisierung
  - Bekannte Implementierungen
    - Apache JDO
    - DataNucleus

Praxis

- Migrationsanalyse
- Zielsetzung
- Durchführung
- Leistungsbetrachtung
- Codevergleich

Praxis

- **Migrationsanalyse**
  - Problemstellen identifizieren und mögliche Workarounds finden
  - An leistungskritischen Prozessen implementieren, um Erfahrungen zu sammeln
  - Vorgehen für die Durchführung erstellen, an der sich andere Entwickler orientieren können
  - Entscheidungshilfe für oder gegen eine komplette Migration der ausgewählten Anwendung

Praxis

- **Zielsetzung**
  - Festlegen von Mindestvoraussetzungen
  - Konkret
    - Ressourcenverbrauch bei kritischen Prozessen darf maximal um 20% steigen
    - Verarbeitungszeit bei kritischen Prozessen darf maximal um 20% steigen
    - Verwendetes ORM-Produkt ist Hibernate

Praxis

- **Durchführung**
  - Probleme die während der Durchführung aufgetreten sind
    - Verwendung primitiver Datentypen
    - Annotationen an den Getter-Methoden und nicht an den Datenfeldzuweisungen
    - Logik in den Getter- und Setter-Methoden
    - Sicherheitsfrage, wenn Annotationen im Code verwendet werden
    - Automatisches Nachladen von Daten in einer Mehrschicht-Architektur führt zu Lazy-Loading-Problem

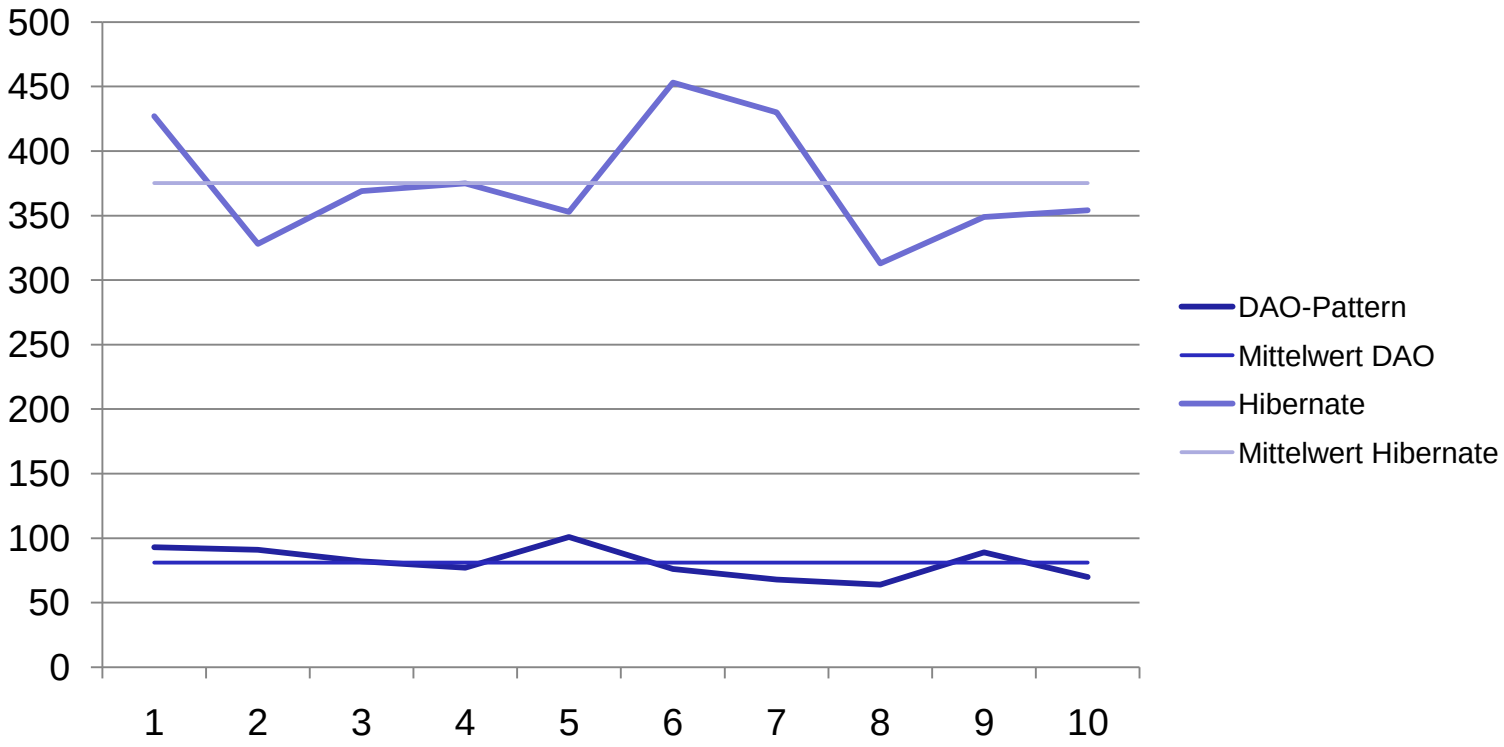
Praxis

- **Leistungsbetrachtung**
  - Vergleich der vorhandenen DAO und der Hibernate Implementierung
  - Wichtig sind möglichst identische Voraussetzungen für die Messungen
  - Aufbau der Testmessungen
    - Daten aus 15 Tabellen laden und an den Client transportieren
    - Der Testfall sollte ein aufwändiger Prozess sein, um Ausmaß bei Problemfällen besser einschätzen zu können
    - Messung umfasste 10 Einzelmessungen und wurden im Client unter der Verwendung des gleichen Aufrufes gemessen
    - Programminterne Zwischenspeicher wurden deaktiviert
    - Dauer zwischen den Messungen mind. 30 Sekunden, um Zwischenspeicherung der SQL-Befehle zu minimieren

Praxis

- **Leistungsbetrachtung - Messreihe**

Messreihe	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



Praxis

- **Codevergleich**
  - DAO Pattern

```
Wohnung whg = null;

PreparedStatement ps = con.prepareStatement("SELECT * FROM WOHNUNG WHERE ID = ?");
ps.setLong(1, wohnungId);
ResultSet rs = (ResultSet) ps.executeQuery();

if (rs.next()) {
    whg = new Wohnung();
    whg.setBezeichnung(rs.getString("BEZEICHUNG"));
    whg.setGewerbeNummer(rs.getInt("GEWERBENUMMER"));
    whg.setSonstigeNummer(rs.getString("SONSTIGENUMMER"));
    whg.setWohnungsNummer(rs.getInt("WOHNUNGSNUMMER"));
}
return whg;
```

  - Hibernate

```
return (Wohnung) entityManager.createQuery("from Wohnung whg where whg.objectId = :objId")
                               .setParameter("objId", wohnungId)
                               .getSingleResult();
```

  - ➔ Einsparung bis zu 75% der Datenzugriffsschicht

Fazit

- **Empfehlungen**
  - Migration der Datenbankzugriffsschicht unter bestimmten Voraussetzungen möglich
  - Auftretende Probleme sind lösbar, bis auf die Leistungsunterschiede
  - Grundvoraussetzung sollte sein, dass der Datenbankzugriff in einer eigenen Schicht erfolgt
  - Vorhandenes DAO-Pattern ließe sich schon mit Hibernate kombinieren

Fazit

- **Erfahrungen**
  - Migration zum ORM-Ansatz muss für jedes Projekt differenziert betrachtet werden
  - Beim Zugriff auf komplexe Tabellenstrukturen und beim Laden großer Datenmengen ist zum Teil sehr lange Zugriffszeiten unvermeidbar
  - Problematisch kann dies in leistungskritischen Prozessen oder während der Batch-Verarbeitung sein

Fragen?

Kontakt

- **Christian Böhmer**

  Software Architekt

  E-Mail: c.boehmer@isys-software.de

- **iSYS Software GmbH**

  Lucile-Grahn-Str. 37

  81675 München

- **Mitarbeiter**

  75, davon 50 Festangestellt