holisticon
MEMBER OF SIGMA GROUP

Ausgemustert? Der Einfluss von EJB 3.0 auf J2EE Design Patterns

Java Forum Stuttgart, 5. Juli 2007

Stefan M. Heldt

Motivation des Vortrags

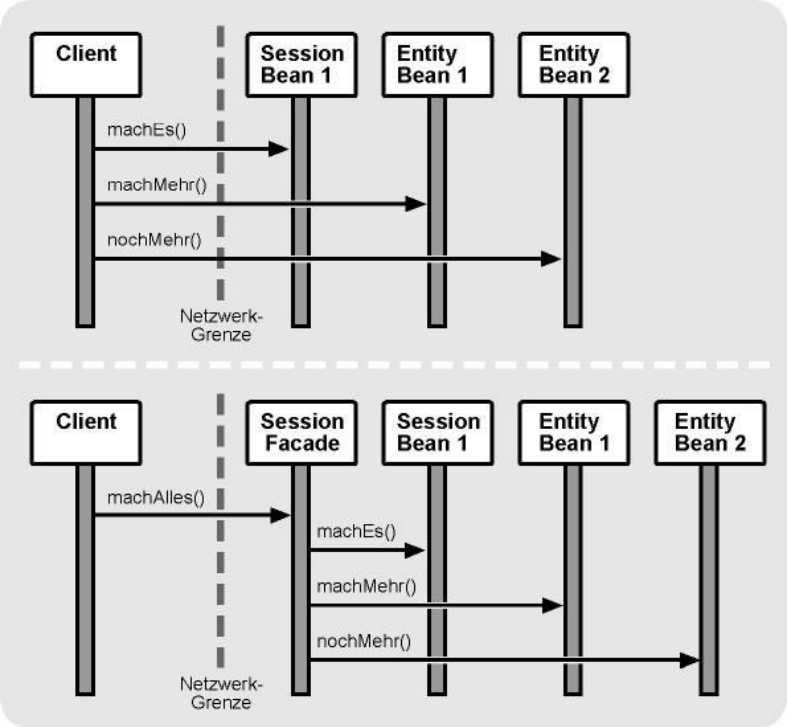
- Design Patterns in der Softwareentwicklung
  - Problembeschreibung
  - Lösungsszenario
- Ease of Development
  - Ziel von Java EE 5.0 und EJB 3.0
- Werden Design Patterns dadurch überflüssig?
- Betrachtung von
  - J2EE Patterns Catalog, Sun
  - EJB Design Patterns, Floyd Marinescu
- Bewertung der zukünftigen Bedeutung

Business Delegate

- Entkopplung des Clients von der Technologie
  - Kapseln von Remote Exceptions
  - Verbergen von Interface-Abhängigkeiten
- EJB 3.0 Business Interface
  - Wirft keine Remote Exceptions
  - bzw. Kapselung durch Runtime Exceptions
  - hat keine technologiebedingte Interface-Abhängigkeit
- Einzige Technologie-Abhängigkeit: Annotationen
  - für Client transparent
- Business Delegate hat ausgedient
  - Ausnahme:
    Sie betrachten Annotationen als Technologieabhängigkeit

Session Facade

- Kapselung entfernter Zugriffe durch Session Beans
  - Reduzierung des kommunikationsbedingten Overheads
  - Kein direkter Zugriff auf Entity Beans
- EJB 3.0 Persistent Entities
  - haben keine Remote-Schnittstelle
  - Werden Clients nur als Detached Object zur Verfügung gestellt
- Session Facade wird durch EJB 3.0 erzwungen



Message Facade / Service Activator

- Asynchrone Kommunikation mit JMS
  - Client muss nicht auf Ausführung warten
  - Client kann kein direktes Ergebnis erhalten
- Message Facade: Message-Driven Bean
- Service Activator: Plain Message Listener
- Wichtigster Aspekt:
  Entkopplung von Client und Server
- Bedeutung beider Design Patterns unverändert

EJB Command

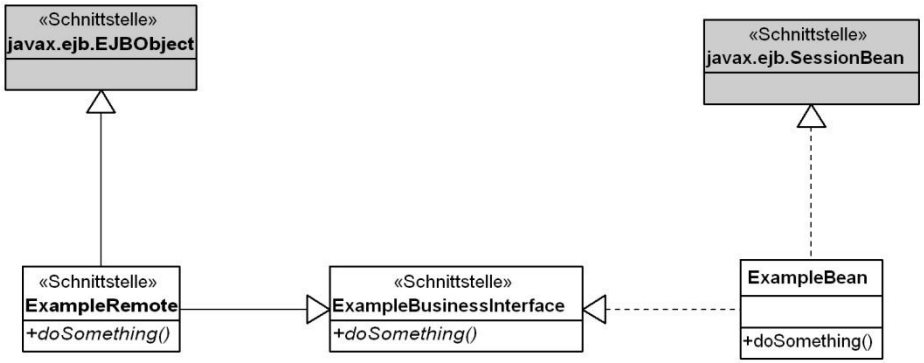
- Selbe Schnittstelle für alle Session Beans
- Analog zum *Command* Pattern von Gamma et. al.
- Einfache Schnittstelle
  - Veränderungen in der Logik nach Außen nicht sichtbar
  - Hinzufügen neuer Session Beans erfordert keine neue Schnittstelle für den Client
- Session Beans sind weiterhin DIE Schnittstelle nach außen
- Bedeutung des Patterns unverändert

EJB Home Factory / Service Locator

- Kapselung des Anforderns von Referenzen auf entfernte Objekte
- EJB Home Factory
  - Spezifisch für Home Interfaces von EJBs
  - Diese gibt es mit EJB 3.0 nicht mehr
  - Daher hat dieses Pattern ausgedient
  - Ausnahme: Verwendung von EJB 2.1 Komponenten
- Service Locator
  - Allgemein für entfernte Objekte
  - Referenzerlangung wird einfacher, da Narrowing entfällt
  - Dennoch sollte dieser technische Ablauf weiter gekapselt werden
  - Bedeutung dieses Patterns unverändert

Business Interface

- Sicherstellung der Konsistenz zwischen
  - Component Interface
  - Bean-Klasse



- Keine Beziehung zwischen beiden Artefakten im Quellcode
- EJB 3.0
  - Business Interface ersetzt Component Interface
  - Bean-Klasse kann und sollte Business Interface implementieren
- Das Pattern Business Interface hat ausgedient

Data Transfer Object / DTO Factory

- Attribute von Entity Beans werden in speziellen Objekten vom/zum Client transportiert
  - Direkten Zugriff auf Entity Beans gibt man dem Client nicht
  - Kapselung durch Session Facade
- Erzeugung der Transportobjekte
- EJB 3.0 Persistent Entities
  - Können als Detached Objects zum Client transportiert werden
  - Persistenzmechanismen im Entity Manager
- Beide Patterns haben ausgedient
- Ausnahme: Custom DTOs
  - Untermenge von Attributen einer Persistent Entity
  - Oft ist die Anpassung des Objektmodells vorzuziehen

Data Transfer Hash Map

- Generisches Data Transfer Object (DTO)
  - Speicherung von Attributen als Key/Value-Paare
  - Generische Implementierung zum Schreiben und Lesen möglich
  - Client muss keine spezifischen DTO-Klassen kennen
- Prinzipiell wie DTO zu bewerten
- Aber:
  - Persistent Entities/Detached Objects in spezifischen Klassen implementiert
  - Generischer Aspekt kann für Clients vorteilhaft sein
- Pattern mit angepasster Motivation weiterhin von Bedeutung

Value List Handler

- Ausschnitte von Entity-Listen
  - Zwischenspeichern der Gesamtliste
  - Seitenweise Navigation durch Gesamtliste
  - Vermeidet unnötigen Datentransport zum Client
- EJB 3.0 Query API erlaubt Segmentierung von Abfragen
- Value List Handler weiterhin notwendig
  - Implementiert Navigationslogik
  - Zwischenspeichern der Gesamtliste nicht mehr notwendig

Generic Attribute Access

- Ähnlich der Data Transfer Hash Map
  - Speicherung von Attributen als Key/Value-Paare
  - Generische Implementierung zum Schreiben und Lesen möglich
  - Client muss keine spezifischen DTO-Klassen kennen
- Aber: Befüllung der HashMap durch die Entity Bean
- Stammt aus der Zeit ohne Local Interfaces
  - Kein Zugriff auf einzelne Attribute von außen
- EJB 3.0
  - Entfernte Zugriffe auf Persistent Entities nicht möglich
  - Notwendigkeit für Befüllung der HashMap in Entity entfällt
- Pattern hat ausgedient

Data Transfer Row Set

- Verzicht auf Entitäten
- Optimiert für Tabellendarstellung
- Nicht objektorientiert
- Direkter Datenbankzugriff
- Verwendung von javax.sql.RowSet
  - Sehr nahe an der Datenbank
  - Als *disconnected row set aber* keine Verbindung zur Datenbank
- Motivation für Pattern mit EJB 3.0 unverändert
- Pattern weiterhin von Bedeutung

Composite Entity

- Entity Beans als grobgranulare Entitäten
- Dependent Objects als feingranulare Entitäten
- Reduzierter Deklarationsaufwand
  - Weniger Entity Beans
  - Weniger Beziehungen
- EJB 3.0 Persistent Entities erlauben
  - Einfache Deklaration von Entities
  - Vererbung
  - Einfache Deklaration von Beziehungen
- Persistent Entities grob- und feingranular
- Pattern hat ausgedient

Dual Persistent Entity Bean

- Eine Entity Bean für
  - Container Managed Persistence (CMP) und
  - Bean Managed Persistence (BMP)
- Abstrakte Oberklasse für CMP
- Konkrete Unterklasse für BMP
- Wechsel CMP/BMP im Deployment Descriptor
- Mit EJB 3.0 gibt es weder CMP noch BMP
- Persistenzmechanismen im Entity Manager
- Pattern hat ausgedient

Data Access Command Bean / DAO

- Kapselung der Zugriffe auf Persistenzschicht
- Keine Verwendung von Entity Beans z.B. wegen
  - Vererbung
  - Multi Table Mapping
- Viele Unzulänglichkeiten von Entity Beans sind mit EJB 3.0 abgestellt
- Zugriffe auf Persistenzschicht mit Entity Manager
- Kapselung dieser Aufrufe in Data Access Command Bean/DAO
- Patterns haben an Bedeutung gewonnen, da sie auch für Persistent Entities sinnvoll sind

JDBC for Reading / Fast Lane Reader

- Verzicht auf Entity Beans
- Vermeidung von komplexen Objektmodellen für lesenden Zugriff
- Bessere Performance durch JDBC-Zugriff auf Datenbank
- Komplexität hat mit EJB 3.0 abgenommen
- Dennoch komplexer als JDBC
- Patterns weiterhin von Bedeutung

Version Number

- Lost Update Problem bei Nebenläufigkeit
  - Mehrere Clients erhalten Kopie der selben Entität
  - Kopien werden geändert und an Server zurückgesendet
  - Änderung der letzten Kopie gewinnt
- Hinzufügen einer Version Number als Attribut
- Vor Update wird Version Number geprüft
- Version Number wird bei Update hochgezählt
- Pattern in Spezifikation übernommen
- @Version markiert Version Number
- Keine Implementierung durch Entwickler

Generierung von Primärschlüsseln

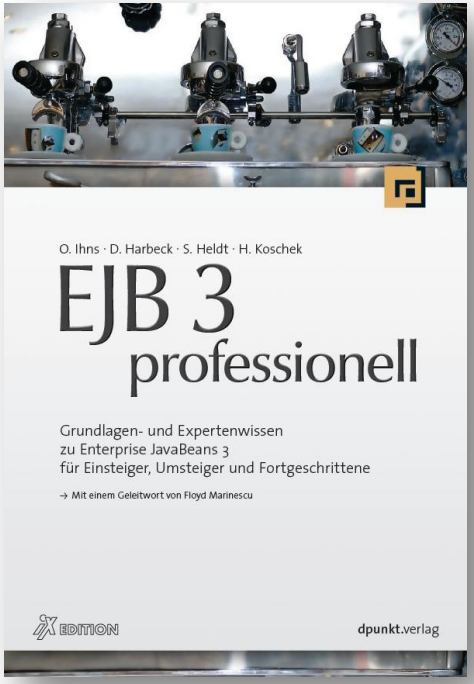
- Bis EJB 2.1 keine Vorgaben zur Generierung von Primärschlüsseln in Spezifikation
- Patterns beschreiben Generierung eindeutiger Schlüssel in der EJB-Schicht
  - Sequence Blocks
  - UUID for EJB
  - Stored Procedures for Autogenerated Keys
- Generierung in Spezifikation übernommen
- Entwickler annotiert
- Container-Hersteller implementiert
- Patterns haben für Persistent Entities ausgedient

Fazit und Ausblick

- Knapp die Hälfte der Patterns sind für Entwickler obsolet
- Einige davon wurden in die Spezifikation übernommen

- Entstehen neue Patterns durch EJB 3.0?
- Wie migriere ich von EJB 2.x nach 3.0?

- Mehr Details im Buch
- Vorabversion am dpunkt-Stand



Vielen Dank
für Ihre
Aufmerksamkeit

stefan.heldt@holisticon.de