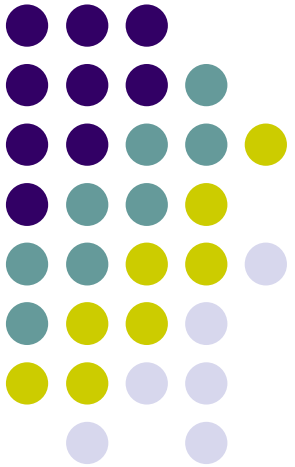
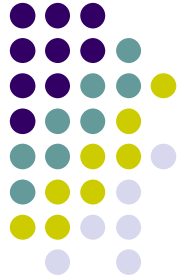
Verteidigung Bakkalaureatsarbeit

Beispielkatalog für EJB-Entwurfsmuster

Ronny Klinger

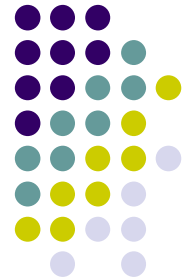


Aufgabenstellung



- Einarbeitung in existierende EJB-Entwurfsmuster
- Aufarbeitung des Hotelbeispiels aus [Cheesman] unabhängig von konkreter Technologie
- Anpassung und prototypische Realisierung des Beispiels unter Verwendung der EJB 2.0-Technologie, dabei Anwendung der EJB-Entwurfsmuster
- Erstellung eines Kataloges übersichtlicher Java-Beispiele für ausgewählte Muster basierend auf der Implementierung des Hotelbeispiels

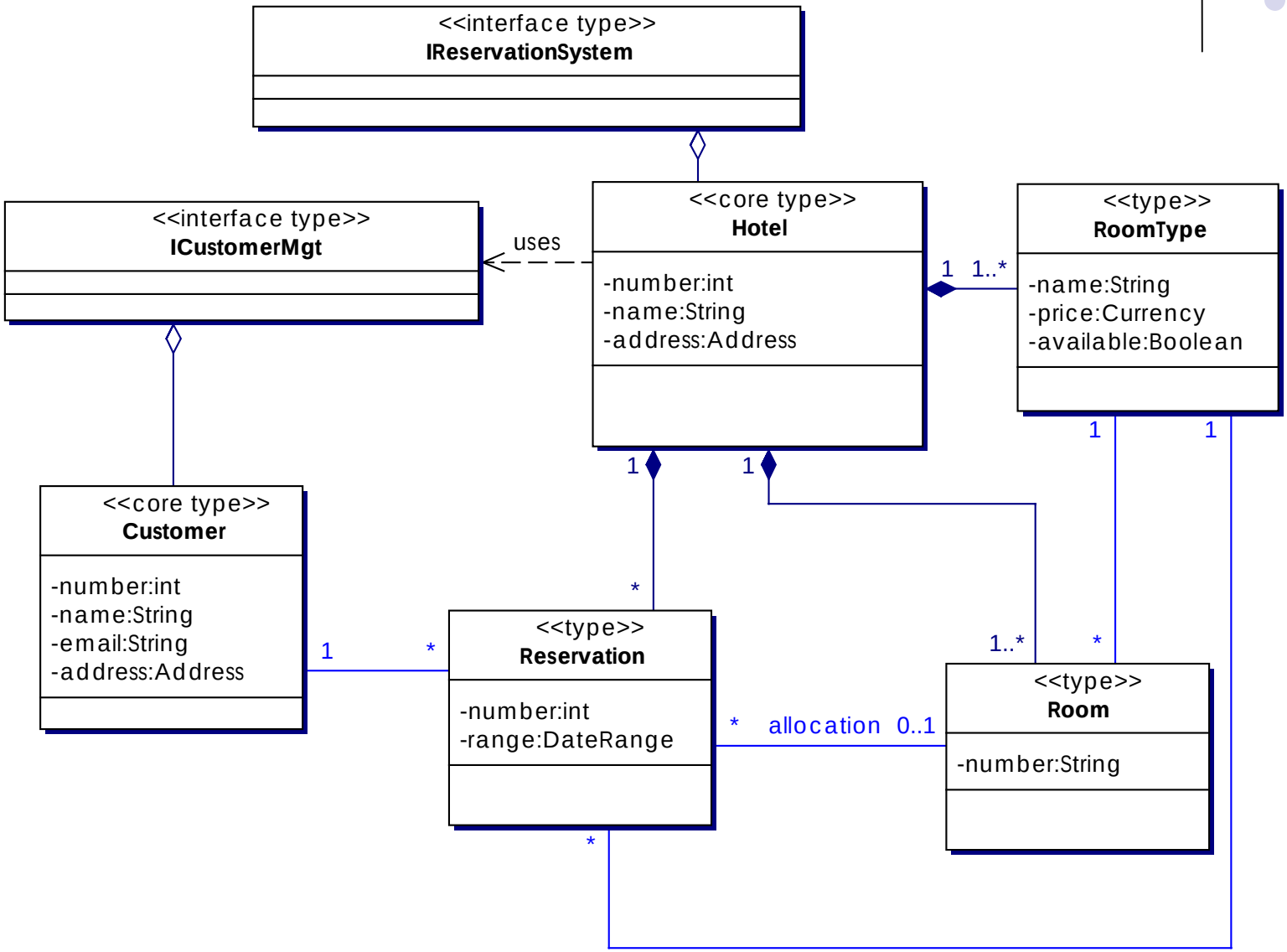
Wichtige EJB-Entwurfsmuster



- existierende EJB-Entwurfsmuster
  - Service Locator
  - **Business Delegate**
  - **Session Façade**
  - **Value Object**
  - Value Object Assembler
  - Value List Handler
  - Aggregate Entity
  - Data Access Object
  - **Service Activator**

im Zwischenbericht detailliert vorgestellt

Systemspezifikation



Anpassung an EJB 2.0

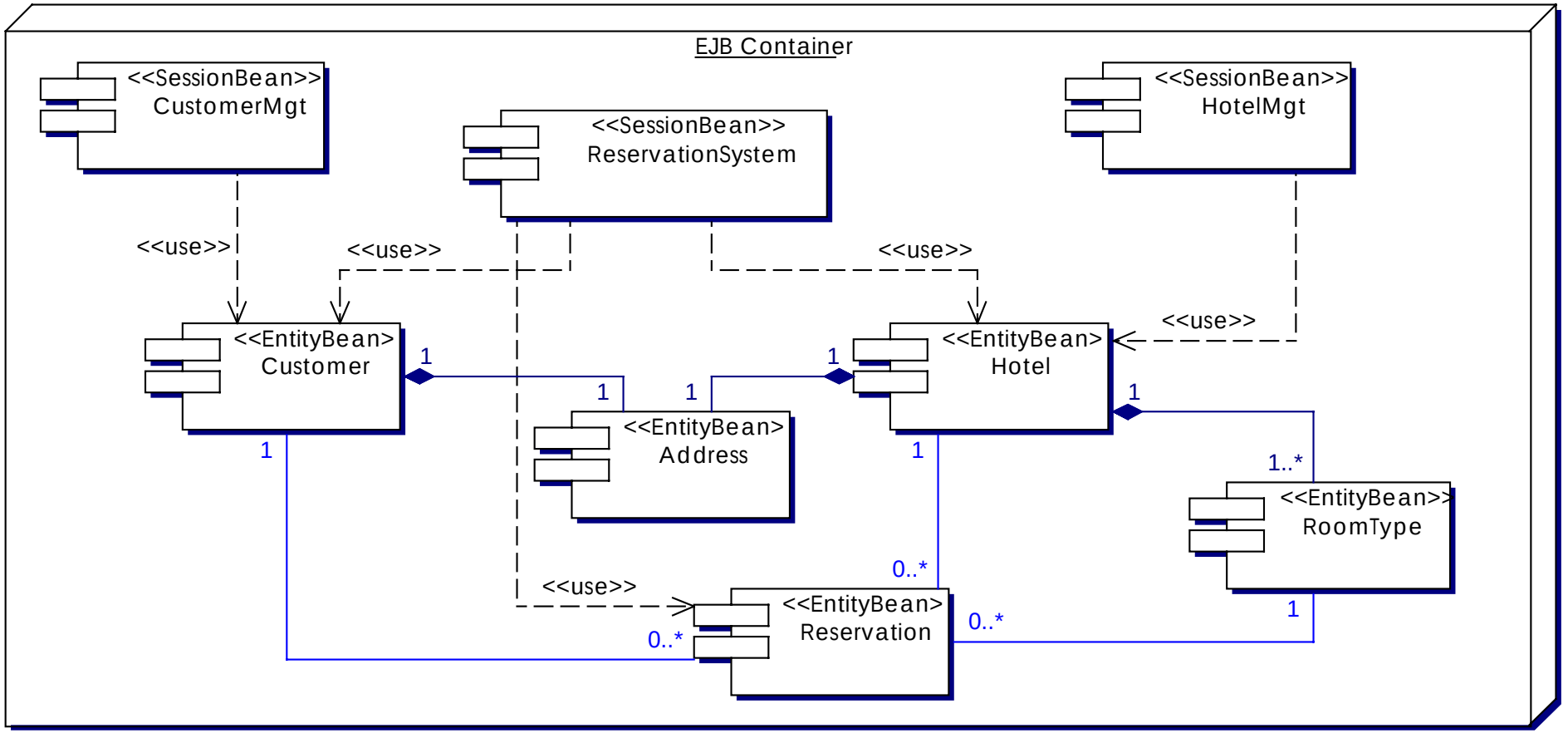
- Anpassung des unabhängig spezifizierten Systems an EJB 2.0
  - Welche Daten sind dauerhaft im System zu speichern?

  →Entity Beans *Customer*, *Hotel*, *RoomType*, Reservation, Address

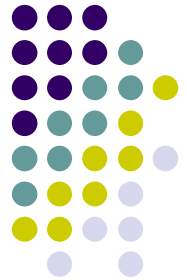
  - Wie wird auf die Daten zugegriffen?

  →Session Beans *ReservationSystem*, *CustomerMgt*, *HotelMgt* als einheitliche Zugriffsschnittstelle

Systemspezifikation mit Enterprise JavaBeans 2.0

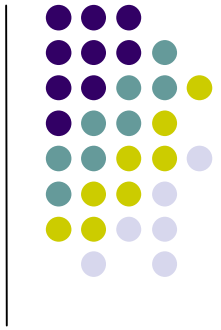


Beispielkatalog von EJB-Entwurfsmustern



- Ziel:
  - Erstellung eines Kataloges übersichtlicher Beispiele für wichtige EJB-Entwurfsmuster
  - Bezugnahme an die Stelle in der prototypischen Realisierung der Hotel-Anwendung, in der das Entwurfsmuster angewandt wird
  - Java-Quellcode

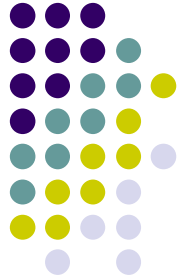
Service Locator (1)



- Aufgaben
  - Speicherung des InitialContext-Objektes und Referenzen auf Home-Interfaces der einzelnen EJBs
- Anwendung
  - immer wenn Zugriff auf Home-Interfaces erfolgt, beispielsweise auf *Address-Bean*
- Implementierung
  - ist selbst ist eine normale Java-Klasse, bei der Entwurfsmuster Singleton angewendet wird

```
public AddressLocalHome lookupAddress() throws NamingException {
   if (objRefAddress==null){
       objRefAddress =AddressLocalHome)initialContext.lookup(
       "java:comp/env/ejb/Address");
   }
   return objRefAddress;
}
```

Service Locator (2)



- Aufwand
  - ohne Service Locator

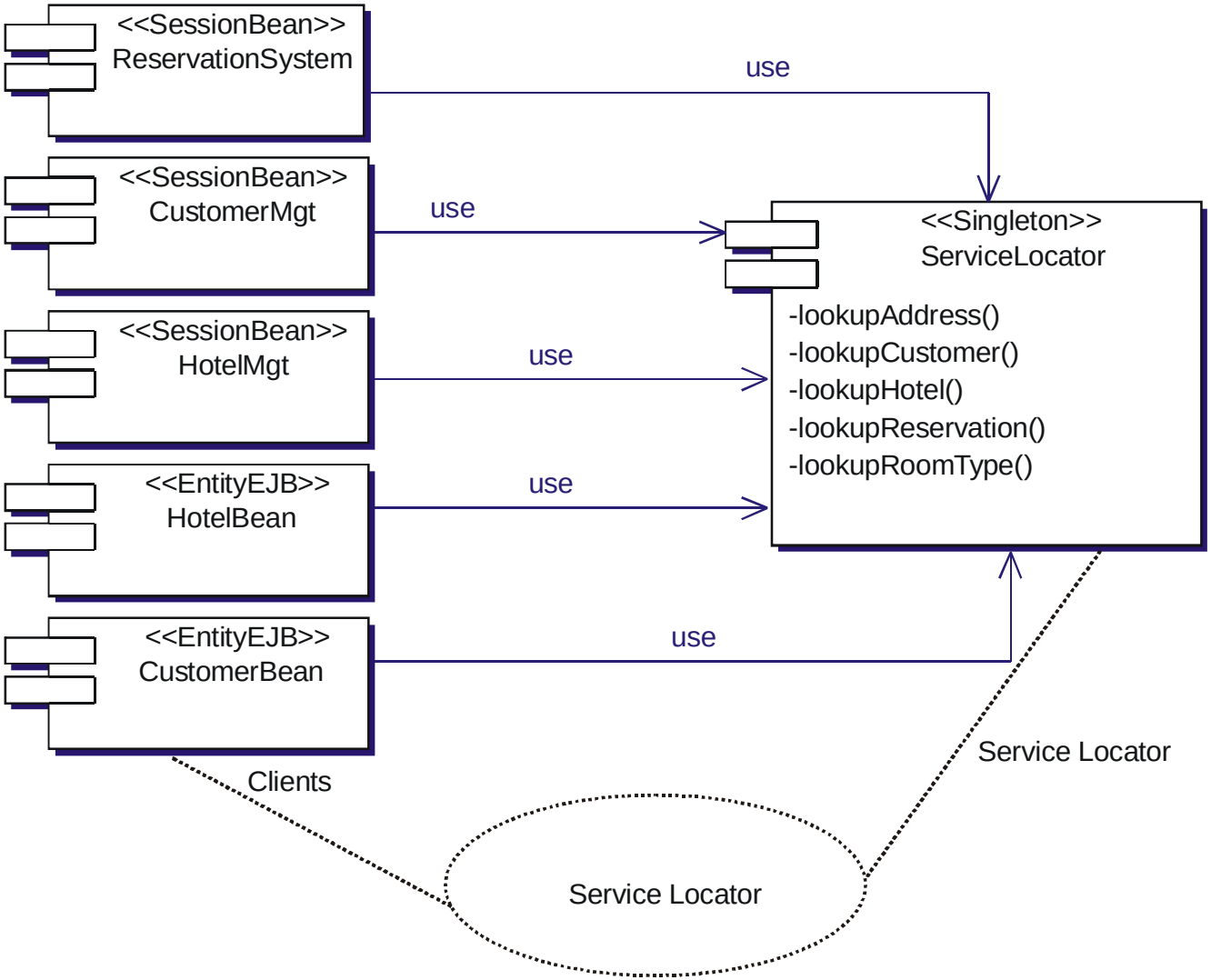
```
AddressLocalHome homeAddress ;
homeAddress =
   (AddressLocalHome)initialContext.lookup(
    "java:comp/env/ejb/Address");
```

  - mit Service Locator

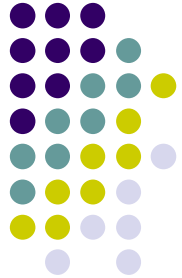
```
AddressLocalHome homeAddress = null;
homeAddress = ServiceLocator.getInstance().lookupAddress();
```

- Vorteil
  - durch Zwischenspeicherung der Home-Interfaces Geschwindigkeitssteigerungen
  - Einsparung unnötiger Zugriffe auf JNDI

Service Locator (3)

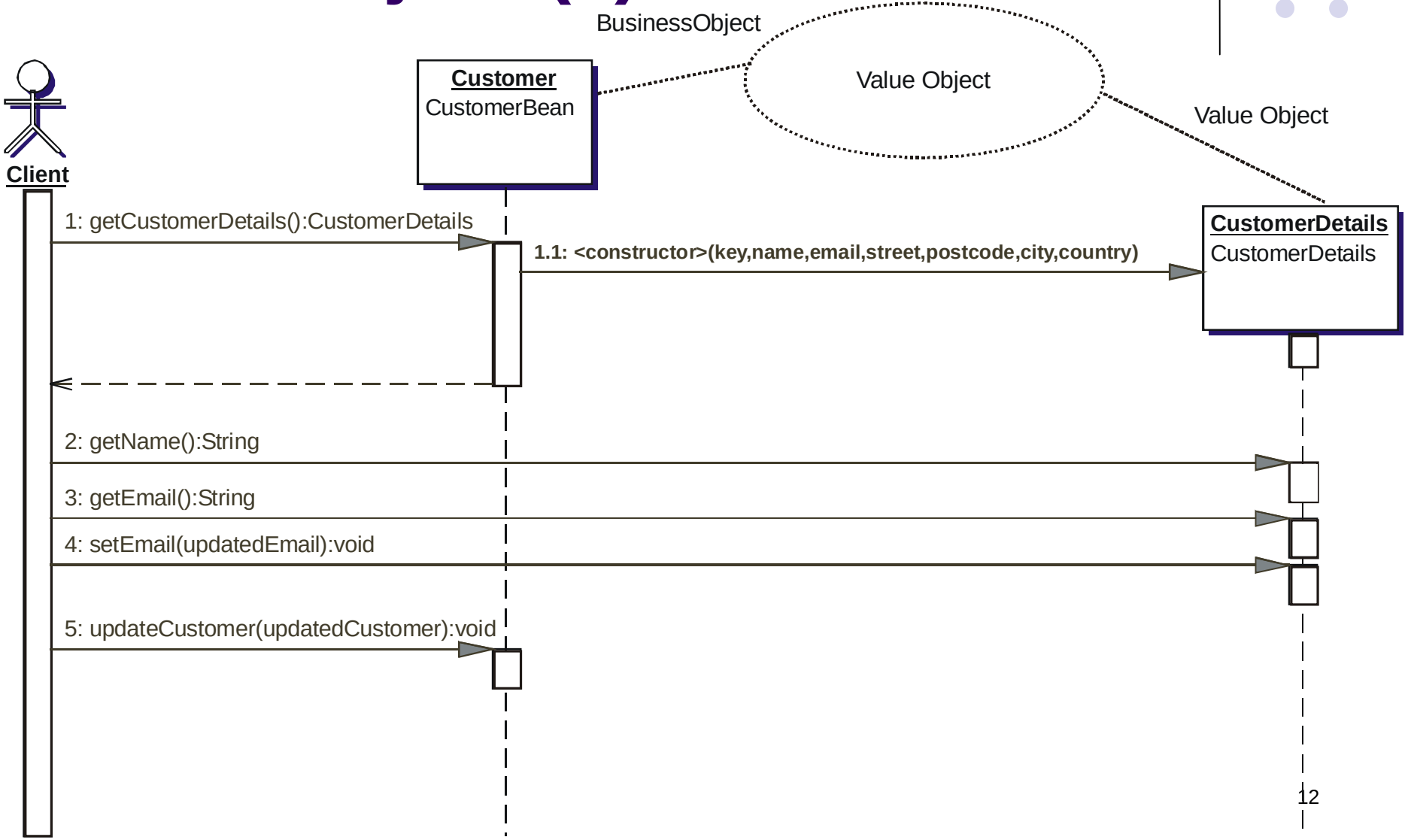


Value Object (1)

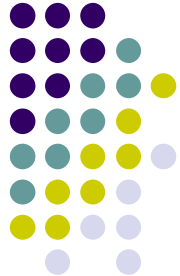


- Aufgaben
  - Reduzierung der entfernten Methodenaufrufe durch Übertragung mehrerer Daten in einem Aufruf
  - Zugriff auf die einzelnen Daten erfolgt über spezielle get- und set-Methoden lokal im Client
  - ebenfalls für EJBs mit lokalen Schnittstellen sinnvoll
- Anwendung
  - Beispielsweise lokales Kunden-Objekt *CustomerDetails* oder Hotel-Objekt *HotelDetails* für Client

Value Object (2)

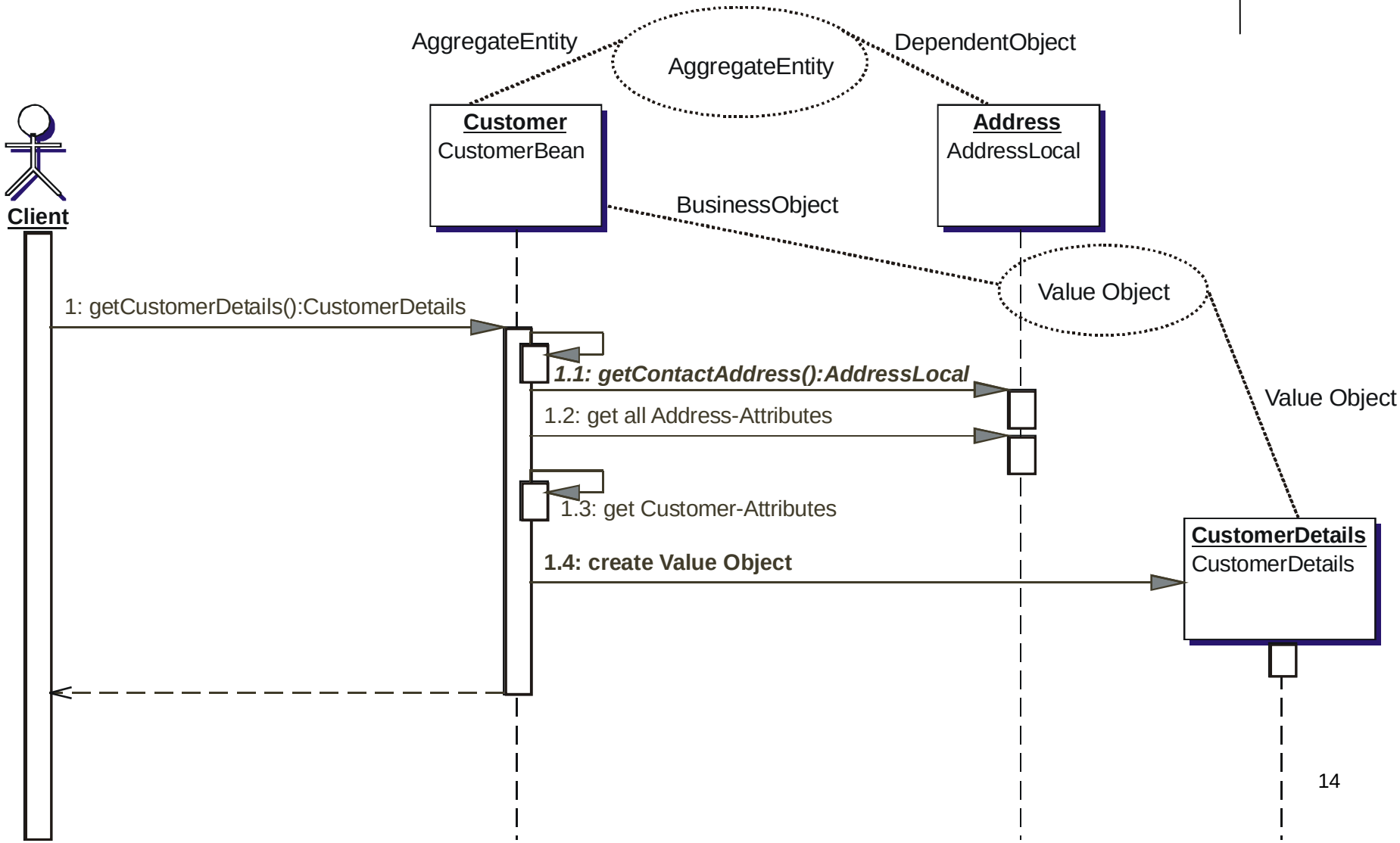


Aggregate Entity (1)

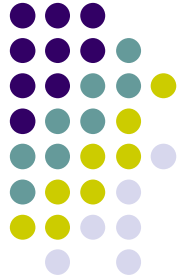


- Aufgaben
  - vereinfachte Handhabung von Objekten, die aus mehreren Entity Beans bestehen
  - unabhängige Bean und beliebig viele abhängige Beans
- Anwendung
  - beispielsweise Kunden-Objekt besteht aus allgemeinen Kundendaten-Bean und Adressen-Bean
  - wird in Verbindung mit Value Object verwendet
  - Relationen zwischen einzelnen Entity Beans werden automatisch durch den EJB-Container verwaltet

Aggregate Entity (2)

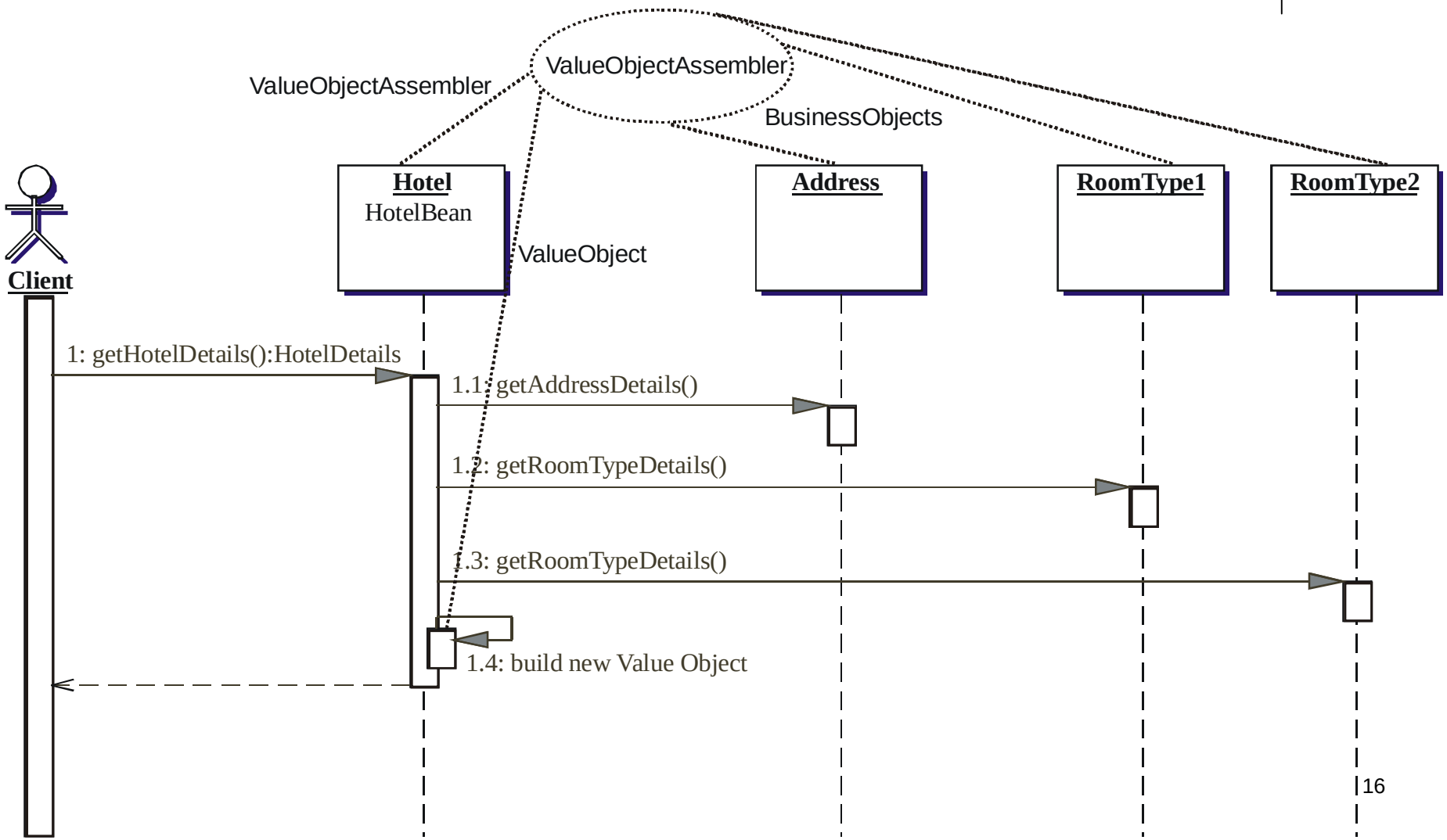


Value Object Assembler (1)

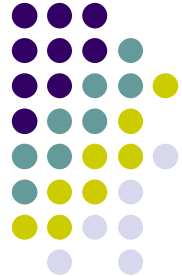


- Aufgaben
  - erstellt aus mehreren Value Objects ein größeres Value Object
- Anwendung
  - beispielsweise Hotel-Objekt *HotelDetails* wird aus Value Objects *AddressDetails* und einer Collection von *RoomTypeDetails* erstellt
  - Value Object sollte auf Grund der hohen Komplexität nicht veränderbar sein

Value Object Assembler (2)

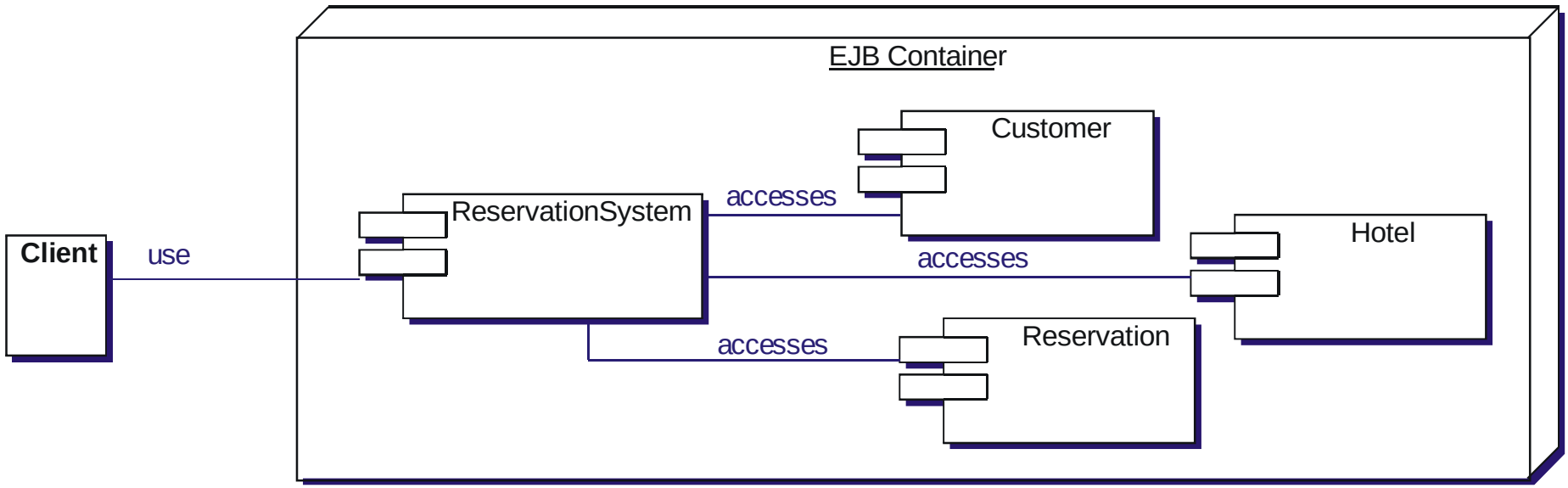


Session Facade (1)

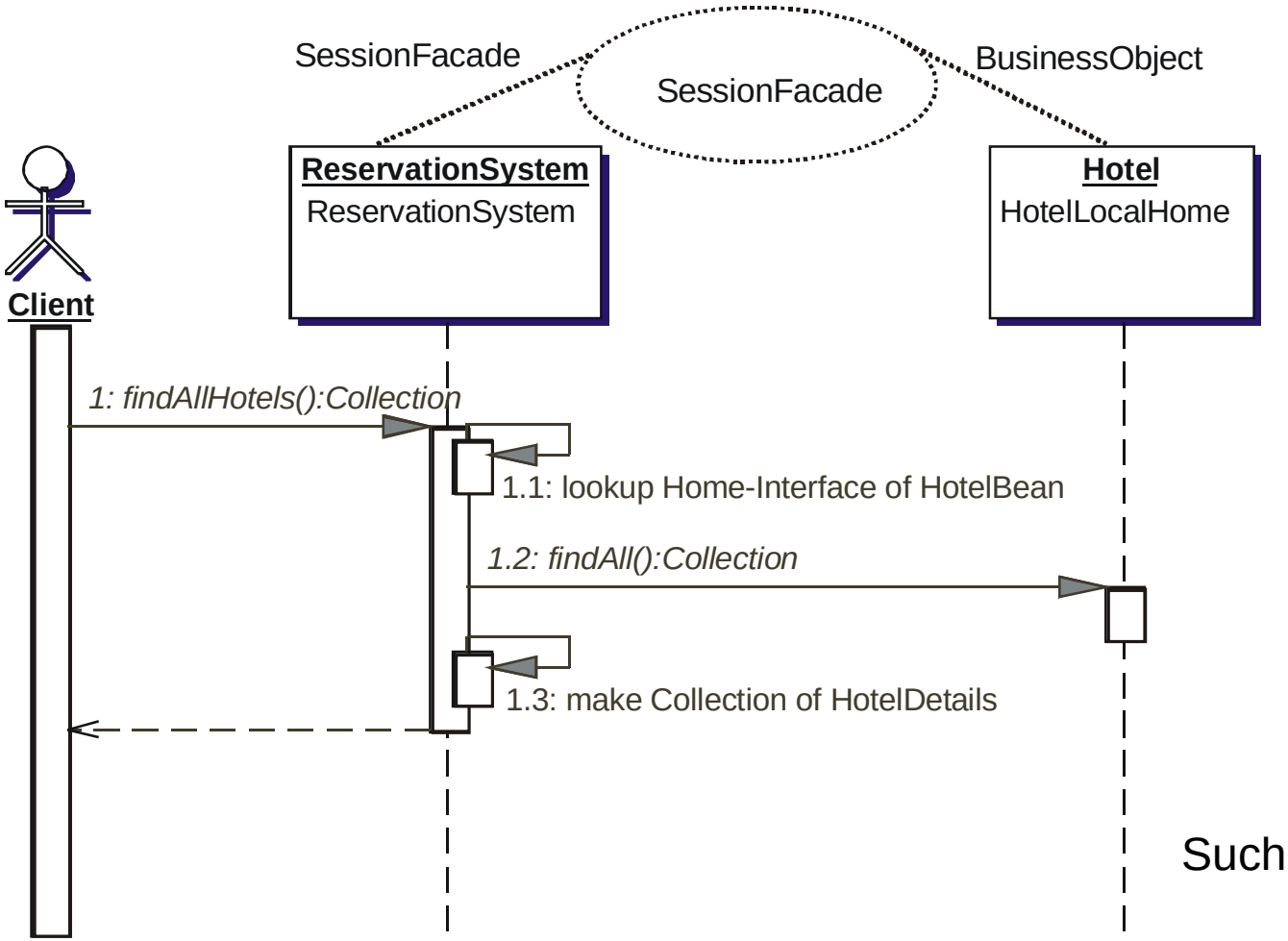


- Aufgaben
  - Realisierung einer einheitlichen Zugriffsschnittstelle für den Client
  - =Abstraktion auf Server-Seite
- Anwendung
  - Zugriffsschnittstelle für das Reservierungssystem, für die Kundenverwaltung und für die Hotelverwaltung

Session Facade (2)

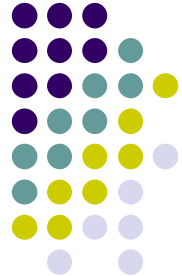


Session Facade (3)



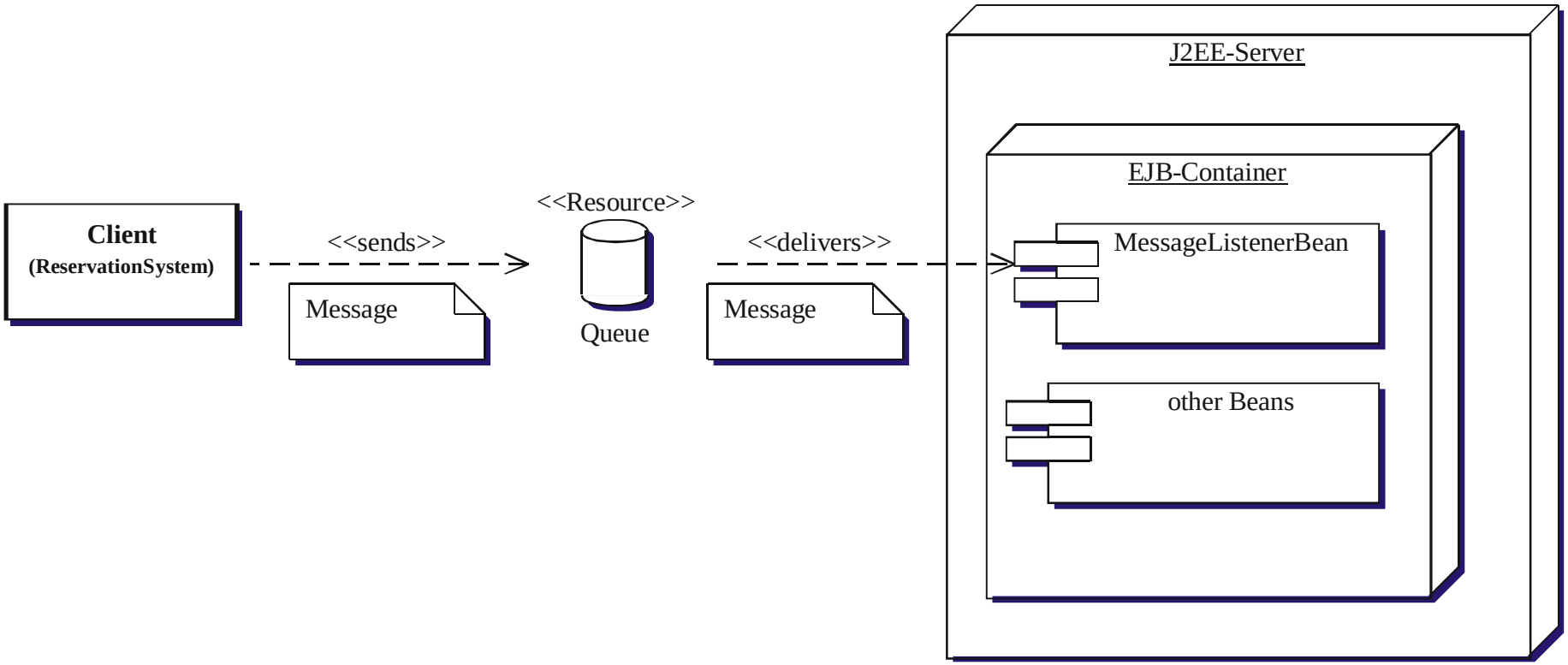
Suche nach allen Hotel

Service Activator (1)

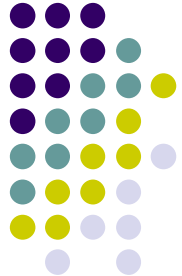


- Aufgaben
  - asynchrone Übermittlung von (Text)nachrichten
- Anwendung
  - Informierung des Abrechnungssystems durch das Reservierungssystem, wenn Kunde seinen Aufenthalt im Hotel beginnt

Service Activator (2)

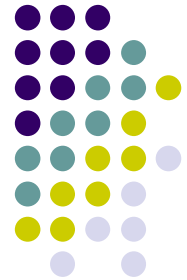


Bewertung



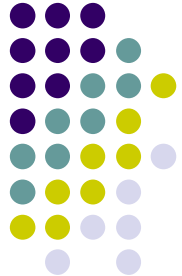
- Vorteile durch die Anwendung von EJB-Entwurfsmuster
  - bessere Übersichtlichkeit des Quellcodes
  - geringere Abhängigkeit zwischen einzelnen Komponenten des Systems
  - teilweise Geschwindigkeitssteigerungen
  - bessere Wartbarkeit der Software
- Nachteile
  - Mehraufwand beim Entwurf und in der Implementierung →hier ist eine Unterstützung durch Entwicklungsumgebungen wünschenswert

Geschwindigkeissteigerungen



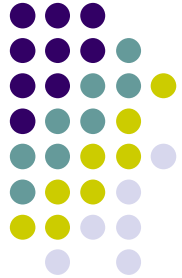
- Service Locator
  - ca. 20% Verbesserung bei 500x Zugriff auf Entity Beans
  - minimaler Mehraufwand von ca. 4 Zeilen Java-Code in Service Locator-Klasse
- Session Facade
  - 20% Verbesserung statt Direktzugriff auf Entity Beans
- Value Object
  - bis zu 650% Verbesserung (bei vielen Attributen)
- Aggregate Entity
  - ebenfalls erhebliche Verbesserungen durch lokale Interfaces

Ausblick



- in Zukunft ist noch mit Verfeinerungen zu rechnen
  - weitere Anpassung an EJB 2.0
  - J2EE-Entwurfsmusterkatalog von Sun befindet sich noch in Entwicklung
- Bessere Unterstützung durch Entwicklungs-umgebungen wünschenswert
  - Generierung einer Value Object-Klasse für Entity Beans
  - Generierung eines Service Locators

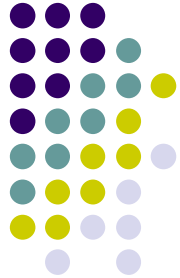
Fazit



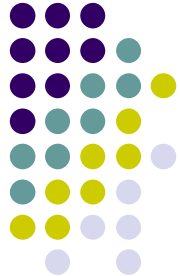
- Verwendung von Entwurfsmustern auf jeden Fall sinnvoll
- schon in Entwurfsphase verschiedene Alternativen in Erwägung ziehen
- Entwurfsmuster nicht isoliert, sondern in Kombination miteinander betrachten

- →zusätzlicher Aufwand für den Entwickler ist in Anbetracht der Vorteile gerechtfertigt

Ende



Literatur



- Cheesman
  - John Cheesman, John Daniels. UML Components A Simple Process for Spezifying Component-Based Systems. Addison-Wesley, Amsterdamm, 2001.