Komponentenbasierte Softwareentwicklung mit PHP

Oliver Schlicht - bitExpert

Überblick

1. Was ist eine Komponente?
2. Entwicklung eines Beispieldesigns
3. Dependency Injection
4. DI Container Garden
5. Demo
6. FAQ
7. Zusammenfassung
8. Ausblick

1. Was ist eine Komponente

- Beispiel:
  - „Wohnraumkomponente“ ;)

1. Was ist eine Komponente?

- Angebote Funktionalität von „Fernseher“
  - ein- und ausschalten
  - Sender wählen

- Funktionalitätsumfang basiert auf dem Interface Fernseher, nicht auf dem eigentlichen Gerät

1. Was ist eine Komponente?

- Benötigte Funktionalität:
  - Strom
  - „Fernsehanschluss“

- es wird nur spezifiziert WAS benötigt wird, nicht WIE das Benötigte arbeitet
- es können verschiedene Geräte an den „Fernseh-anschluss“ angeschlossen werden, solang sie die benötigte Spezifikation erfüllen

1. Was ist eine Komponente?

- Zusammenfassung:
  - Angebotene Funktionalität (Provided Interface, PI)
  - Benötigte Funktionalität (Required Interface, RI)
  - Black Box
  - austauschbar

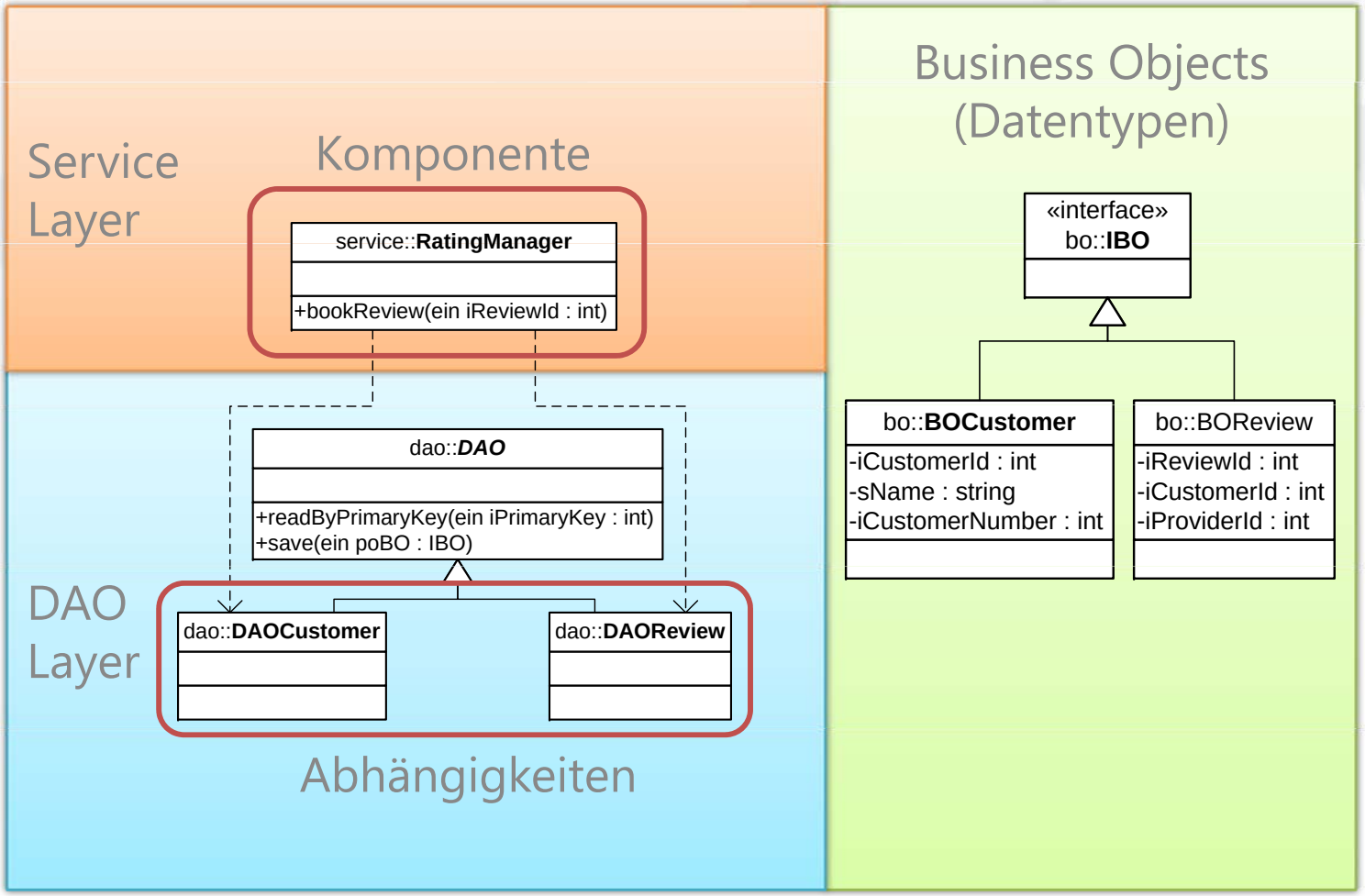
- Übertragung in (PHP)OO-Welt
  - Komponente == Klasse ?
  - Angebotene Funktionalität -> Klassenschnittstelle
  - Benötigte Funktionalität -> ?

2. Entwicklung eines Beispieldesigns

- Ratingsystem - Anforderungen
  - Ausgangspunkt: „Ebay ohne Bewertungssystem“
  - Kunden sollen Anbieter nach erfolgter Geschäftsabwicklung bewerten können
  - Anbieter tragen ein Rating mit sich, welches durch abgegebene Reviews beeinflusst und verändert wird
- Komponente „RatingManager“
- Datentypen: BOCustomer, BOReview
- DAO: DAOCustomer, DAOReview

2. Entwicklung eines Beispieldesigns

- Erster Entwurf

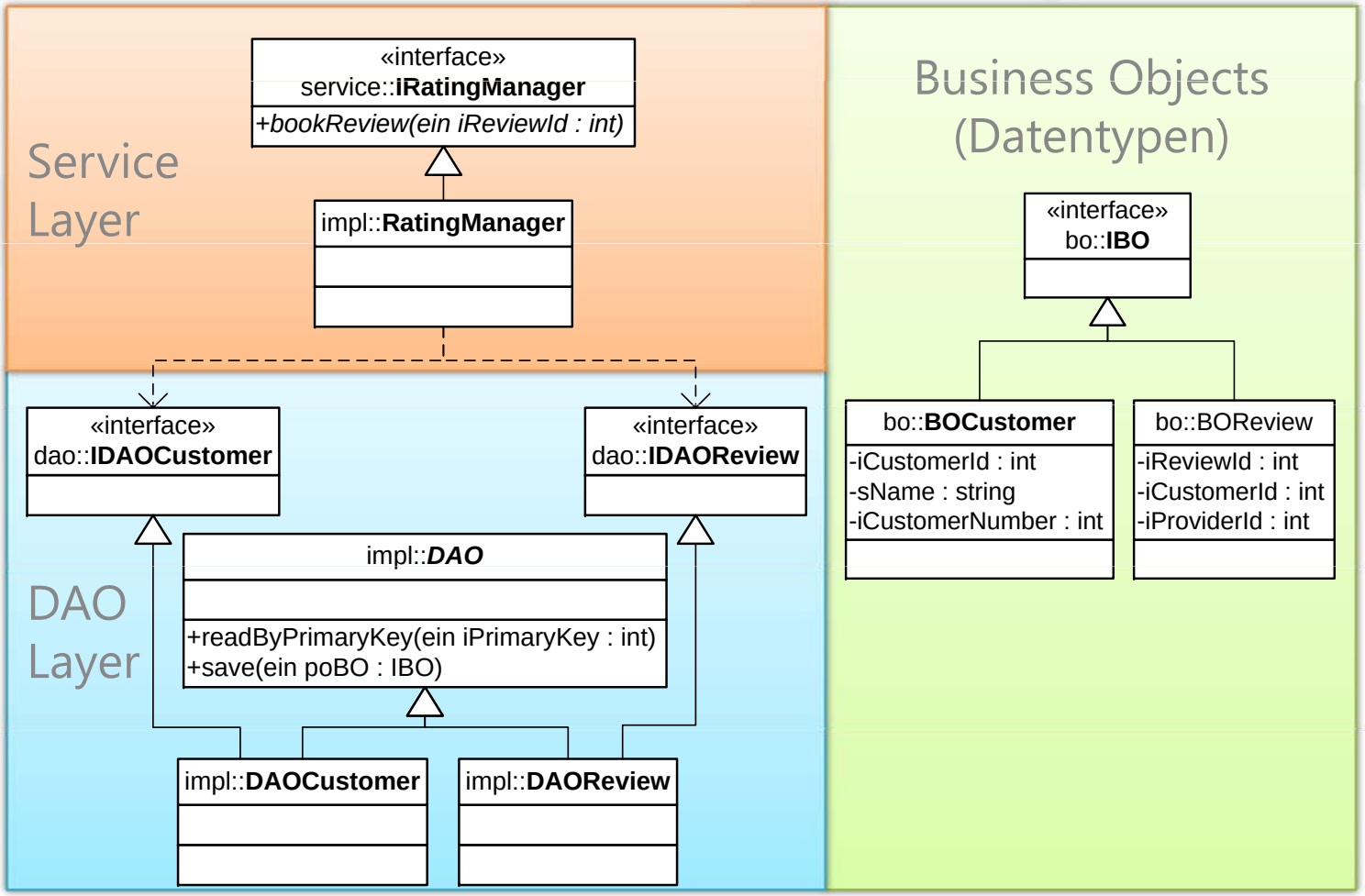


2. Entwicklung eines Beispieldesigns

- Analyse
  - Probleme
    - Kein PI
      - Nutzende Komponenten binden sich direkt an die Implementierung
    - Kein RI
      - Benutzer der Komponente weiß nichts über versteckte Abhängigkeit
    - direkte Kopplung an Implementierung von benötigten Komponenten
      - Feste Bindung sorgt für schlechte Austauschbarkeit und „Untestbarkeit“
- Schritt 1
  - Programmieren gegen Interfaces

2. Entwicklung eines Beispieldesigns

- Interfaces für Komponenten einführen



2. Entwicklung eines Beispieldesigns

- Analyse
  - Probleme
    - Keine PI
      - Nutzende Komponenten binden sich direkt an die Implementierung
    - Kein RI
      - Benutzer der Komponente weiß nichts über versteckte Abhängigkeit
    - direkte Kopplung an Implementierung
      - Feste Bindung sorgt für schlechte Austauschbarkeit und „Untestbarkeit“
    - Wie kommt die Komponente an Instanzen benötigter Klassen?

2. Entwicklung eines Beispieldesigns

- Analyse
  - Probleme
    - Kein RI
      - Benutzer der Komponente weiß nichts über versteckte Abhängigkeit
    - Wie kommt die Komponente an Instanzen benötigter Klassen?

- Schritt 2
  - Dependency Injection

3. Dependency Injection

- Objekt erzeugt Referenzen nicht selbst sondern bekommt sie zugewiesen („Inversion of Control“)
  - Folge: Abhängigkeiten sind an der Schnittstelle erkennbar (RI)
  - Objekt bleibt passiv
  - Referenzen können ausgetauscht werden (bessere Testbarkeit

3. Dependency Injection

- Constructor Injection
  - Vorteil
    - Nach dem Instanziieren der Komponente weiß ich, dass ich ein fertig konfiguriertes Objekt habe
  - Nachteil
    - Im Nachhinein ist keine Umkonfiguration mehr möglich
- Setter Injection
  - Vorteil
    - Komponente wird zur Laufzeit umkonfigurierbar
  - Nachteil
    - evtl. Nullpointer durch nicht initialisierte Referenzen

4. DI Container - Garden

- Problem
  - Wer instanziiert die Komponente bzw. steuert den Lebenszyklus?
  - Woher weiß derjenige, welche Instanz er einsetzen soll?
- Lösung: Container
  - Stück Software, dass Komponenteninstanzen erzeugt, zerstört und Funktionalität für K. anbietet
  - ermöglicht das Eingreifen in den Zyklus über Callbackinterfaces

4. DI Container - Garden

- Garden (http://garden.clawphp.org/)
  - angelehnt an Spring (www.springframework.org)
  - Zugriff auf Komponenten über ApplicationContext
    - im Prinzip eine Factory
  - Beschreibung der Komponenten per XML

4. DI Container - Garden

- Komponentenkonfiguration

```
<beans>

  <!-- Services -->
  <bean id=„RatingManager“ class=„service.RatingManager“>
    <constructor-arg ref=„DAOCustomer“ />
    <constructor-arg ref=„DAOReview“ />
  </bean>

  <!- DAOs -->
  <bean id=„DAOCustomer“ class=„dao.DAOCustomer“>
    ..
  </bean>

  <bean id=„DAOReview“ class=„dao.DAOReview“>
    ..
  </bean>

</beans>
```

4. DI Container - Garden

- Zugriff auf die Komponente

```
<?php

  // Initialisieren des AC
  // XML File wird geparst
  $oCtx = ApplicationContext::getInstance();

  // Komponente holen
  // (wird instanziiert oder aus dem Cache geladen)
  $oRatingManager = $oCtx->getBean(„RatingManager“);

  $oRatingManager->bookReview(1);

?>
```

- -> 5. Demo...

6. FAQ

- Welche Klassen werden Komponenten?
  - Im allgemeinen Klassen die Dienste erbringen
  - Keine Klassen die Datenstrukturen beschreiben
- Entwicklungsaufwand
  - Overhead an XML Beschreibungsdaten
  - Vorteile
    - Klare Trennung von Komponentenschnittstelle, -implementierung und – konfiguration (SOC)
    - Komponente lässt sich durch einfaches Umkonfigurieren mehrfach nutzen

6. FAQ

- Wie groß ist der Parsingaufwand?
  - XML sollte nicht jedes Mal neu geparst werden -> Serialisieren und in Datei legen
  - Aktuelles Projekt (20000 Zeilen Code) -> 10 – 20 ms
- Globale Abhängigkeit vom ApplicationContext
  - ACs sollten möglichst selten instanziiert werden
  - Konfigurationsdaten über „configvalue“ setzen
  - Zugriff auf den AC über IApplicationContextAware

7. Zusammenfassung

- Methodische Aspekte
  - Trennung von Interface und Implementierung
  - Deklarierung von PI und RI (zumindest durch Doku)
- Mit DI / Garden möglich
  - Offenlegung des RI an der Schnittstelle der Klasse
  - Trennung von Implementierung und Konfiguration
  - Bessere (Unit) Testbarkeit der Komponenten
  - Bessere Wartbarkeit / Wiederverwendung

8. Ausblick

- Erweiterte Features des bitFramework
  - DI gesteuertes MVC Framework (ähnlich Spring MVC)
    - ActionResolver, ViewResolver,
    - Validatoren
    - FormActions
  - Vorgefertigte Komponenten / Basisklassen
    - Authentifiziertung, Authorisierung, Transaktionshandling
    - Geschäftsobjekte (BO), DataAccessObjects (DAO)
  - Integration für PHPUnit
    - Unittest, Integrationstest, transaktionale Integrationstests
  - Utilities
    - Mail, Datumsklassen, Logging usw.

8. Fragen?

Quellen

- Martin Fowler – Dependency Injection
  - http://www.martinfowler.com/articles/injection.html
- Steve McConnell – Code Complete
  - http://www.amazon.de/Code-Complete-Deutsche-AusgabeDer-Second/dp/386063593X/sr=8-2/qid=1168434807/ref=pd_ka_2/302-8843512-8120023?ie=UTF8&s=books

- Kontakt:
  - o.schlicht@bitExpert.de