Hochschule Kaiserslautern
University of Applied Sciences

Andreas Eitel
University of Applied Sciences
Kaiserslautern
andreas.eitel@hs-kl.de

SOFTWARE-ARCHITEKTUR:

AUFGABENPAKET 1

6. Februar 2022

Vorbemerkung

Die Hauptthemen dieses Arbeitspakets sind Verteilte Systeme und zugehörige Architekturmuster. Dem Arbeitspaket liegen die Kapitel „Architekturmuster und Design-Patterns“ (bis 4.2.7) und „Verteilte Systeme mit Java-RMI und JMS“ (bis 6.2.3) zugrunde. Bevor Sie die Aufgaben bearbeiten, sollten Sie die beiden Kapitel im Skript durchgearbeitet haben.

Theoretischer Teil

1. Beantworten Sie *kurz* schriftlich:
   (a) (2 Punkte) Nennen Sie zwei wesentliche Begriffe aus dem Bereich Basiskonzepte der Software-Architektur und erklären Sie diese kurz mit eigenen Worten.
   (b) (6 Punkte) Welche Qualitätsattribute – anhand denen man Software-Architekturen bewerten kann – definiert der ISO-Standard 9126? Erklären Sie Qualitätsmerkmale jeweils in 1-2 Sätzen.
   (c) (1 Punkt) In welche Kategorien werden Architekturmuster von Buschmann et al. [BMR+98] eingeteilt?
   (d) (2 Punkte) Nennen Sie beispielhaft zwei Architekturmuster und zwei Design-Patterns die bei verteilten Systemen zum Einsatz kommen.
   (e) (2 Punkte) Was versteht man unter Java RMI? Beschreiben Sie die Funktionsweise/den Ablauf kurz mit eigenen Worten.

Praktischer Teil

Nutzen Sie Eclipse oder IntelliJ zum Bearbeiten der nachfolgenden Aufgaben. Laden Sie Ihre Lösung als ZIP-Datei hoch. Build-Artefakte sollten nicht hochgeladen werden.

2. **Einfaches Java-RMI-Beispiel**
Schreiben Sie eine kleine Client/Server-Anwendung mit Hilfe von Java-RMI. Der Server soll folgende Remote-Schnittstelle anbieten:

```
1 public interface ServerConf extends Remote
2 {
3   public String getDateAndTime(Locale locale) throws RemoteException;
4   public String getOperatingSystem() throws RemoteException;
5   public String getJavaVersion() throws RemoteException;
6 }
```

Listing 1: Remote-Interface des Servers

Über die Methode `getDateAndTime` kann der Client eine Sprachabhängige Datums- und Zeitangabe anfordern. Über die Methoden `getOperatingSystem` und `getJavaVersion` kann das Betriebssystem und die auf dem Server verwendete Java-Version abgefragt werden.

Eine sprachabhängige Datums- und Zeitangabe kann z. B. durch die Zeile `DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat.MEDIUM,locale).format(new Date());` erzeugt werden. Das Betriebssystem kann über `System.getProperty("os.name")` und die Java-Version über `System.getProperty("java.version")` abgefragt werden.

Tipp: Man kann die RMIRegistry (NamingService) auch direkt aus Java, d. h. dem Server-Programm starten. Das macht das Arbeiten in vielen Fällen einfacher. Der Aufruf lautet: `Registry registry = LocateRegistry.createRegistry(Registry.REGISTRY_PORT);`

(a) (4 Punkte) Die Server-Anwendung existiert und funktioniert.

(b) (2 Punkte) Die Client-Anwendung existiert und funktioniert.

3. **Buchlager: Client-/Server-Architektur auf Basis von Java-RMI**
Im Übungspaket finden Sie eine kleine Anwendung mit Swing als GUI-Framework, für eine „Buchlageranwendung“. Mit der Anwendung kann in einem Buchlager nach Büchern gesucht werden. Einziges Suchkriterium ist der Autorname. Die Trefferliste wird angezeigt und man kann sich Detailinformationen von einzelnen Büchern anschauen bzw. Bücher in den Warenkorb legen (vgl. Abb. 1, 2, 3).

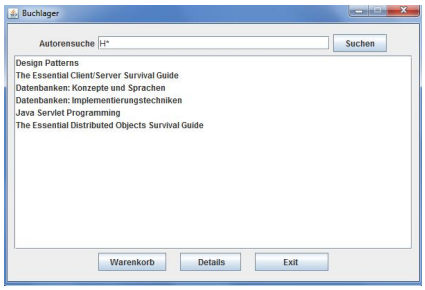

Abbildung 1: Suchseite

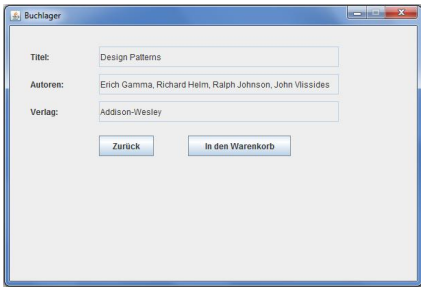

Abbildung 2: Detailanzeige

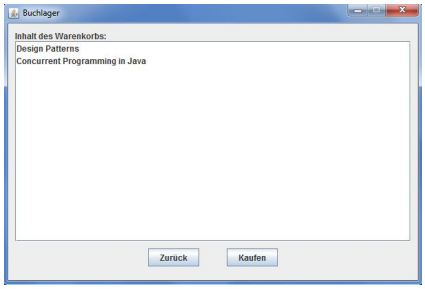

Abbildung 3: Warenkorb

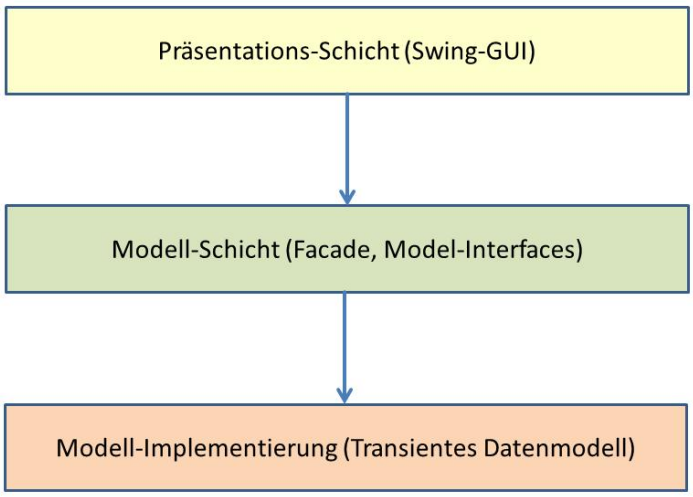

Abbildung 4: Die Schichten der Buchlageranwendung.

Die Anwendung besitzt eine einfache Schichtenstruktur mit den üblichen Schichten: Präsentation, Modell, Datenhaltung (vgl. Abb. 4). Zur Datenhaltung wird keine Datenbank benutzt, sondern die Objekte werden transient im Speicher erzeugt und verwaltet. Abbildung 5 zeigt die Paketstruktur (der Swing-Anwendung). Einstiegspunkt für die Businesslogik ist die Klasse `BuchlagerFacade`.

Die Buchlageranwendung soll nun in eine Client-/Server-Architektur überführt werden. Hierzu wird die Präsentationsschicht abgetrennt und auf einen Client verlagert. Die Modell- und Datenhaltungsschicht bilden den Server. Als Kommunikation soll Java-RMI eingesetzt werden. Abbildung 6 zeigt den schematischen Aufbau.

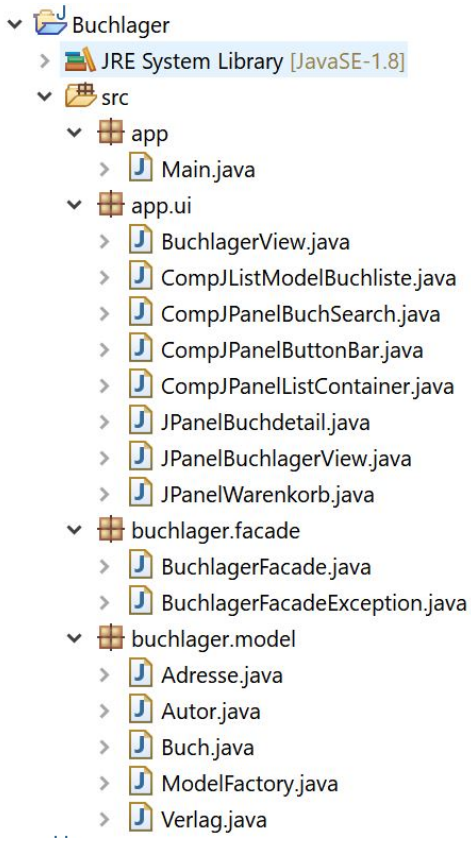

Abbildung 5: Die Paketstruktur der Buchlageranwendung.

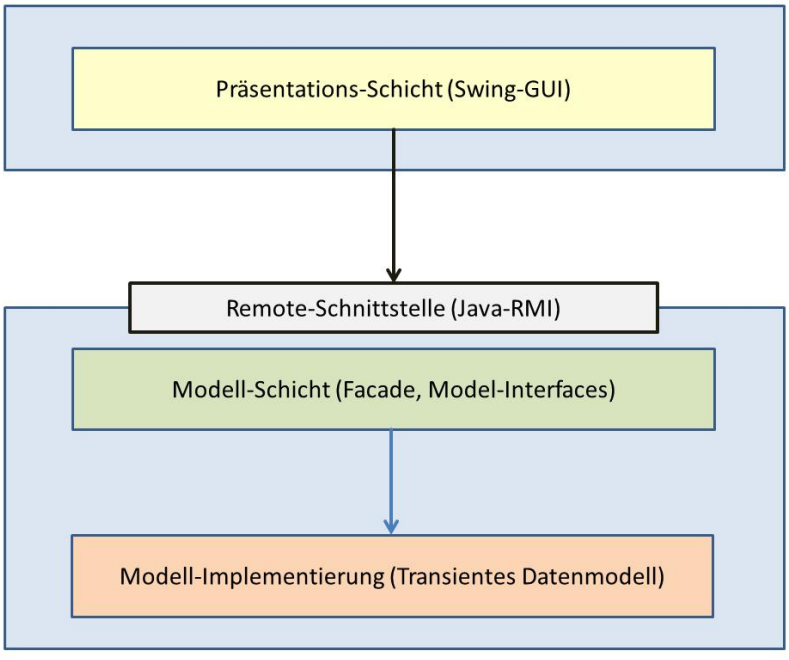

Abbildung 6: Client-/Server-Struktur der Buchlageranwendung.

Die Abtrennung der Präsentationsschicht lässt sich einfach bewerkstelligen, da es nur Abhängigkeiten in Richtung „Modell“ gibt, d. h. es existieren keine Zyklen.

Es kann davon ausgegangen werden, dass sowohl der Client als auch der Server auf demselben Rechner läuft, d. h. die RMIRegistry kann vom Serverprogramm gestartet werden und der Client kann `localhost` für den Zugriff auf die RMI-Registry verwenden.

(a) (1 Punkt) Die Gesamtanwendung funktioniert.

(b) (1 Punkt) Die Auftrennung der Schichten wurde korrekt durchgeführt.

(c) (1 Punkt) Es wurde Maven verwendet, um gemeinsame Klassen an zentraler Stelle zu pflegen.

(d) (3 Punkte) Client-/Server-Implementierung ist korrekt und konkurrierende Zugriffe werden vermieden.

Literatur

[BMR+98] Buschmann, Frank ; Meunier, Regine ; Rohnert, Hans ; Sommerlad, Peter ; Stal, Michael: *Pattern-orientierte Software-Architektur*. Addison-Wesley, 1998