Übungen zur Vorlesung

Softwaretechnologie

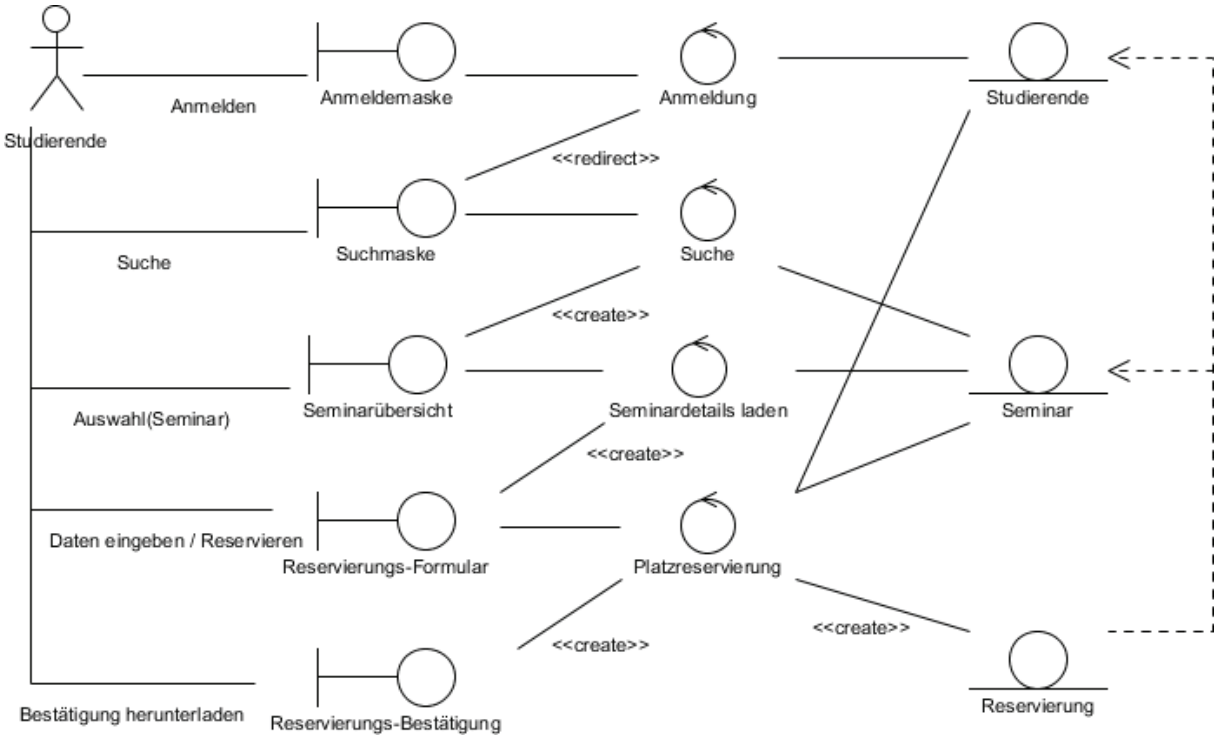
-Wintersemester 2009/2010-

Dr. Günter Kniesel, Pascal Bihler

Übungsblatt 6 - Lösungshilfe

Aufgabe 1. *System-Entwurf* (11 Punkte)

Zu entwerfen ist ein Seminar-Verwaltungssystem vom „Thin-Client“-Typ mit nachfolgendem Analysemodell. Führen Sie, wie in der Vorlesung beschrieben, ein Systementwurf durch. Entscheiden Sie sich im Zweifel für die einfachste Lösung, die funktioniert.

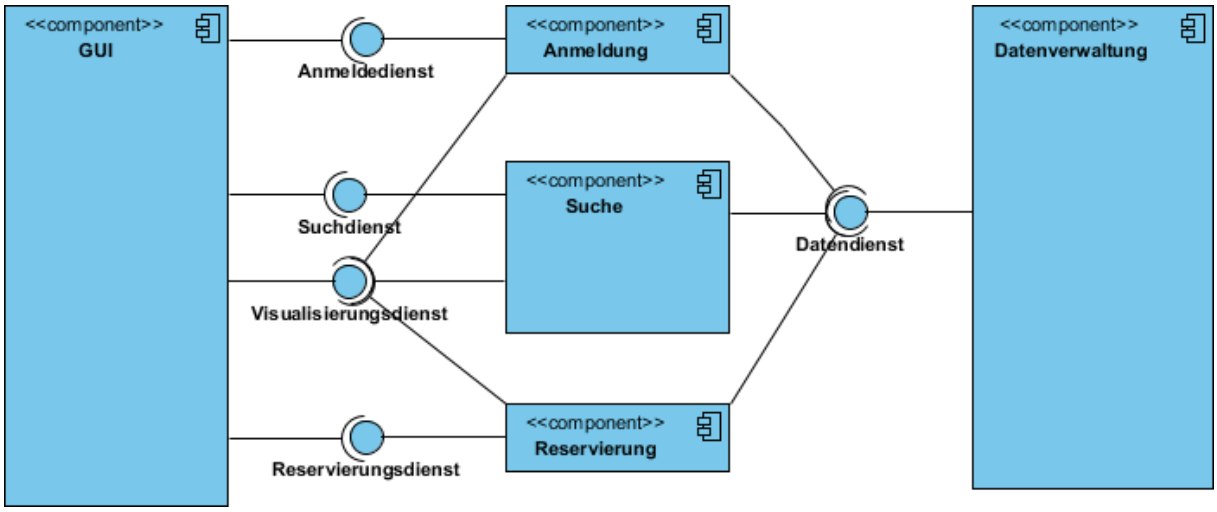


a) Definieren Sie sinnvolle *Dienste* (Interfaces mit komplett typisierten Operationen). Eine textuelle Aufzählung ist für diesen Aufgabenteil ausreichend (kein Diagramm erforderlich).

- Datendienst
  - Klassen: Seminar, Reservierung, Studierende
  - Methoden der Schnittstelle:
    - getSeminare(Suchfilter sf): List <Seminar>
    - getSeminar(int seminarId): Seminar
    - getFreiePlätze(Seminar se): int
    - reserviere(Seminar se, Studierende st)
- Anmeldedienst

    - Klassen: Anmeldung, Studierende
    - Methoden der Schnittstelle:
        - anmelden(String login, String kennwort): Suchmaske
        - abmelden(): Anmeldemaske
- Suchdienst
    - Klassen: Suche, Seminardetails laden, Seminar
    - Methoden der Schnittstelle:
        - getAlleSeminare(): Seminarübersicht
        - getSeminare(Suchfilter sf): Seminarübersicht
        - getSeminardetails(Seminar se): Reservierungs-Formular
- Reservierungsdienst
    - Klassen: Seminarreservierung, Reservierung, Seminar, Studierende
    - Methoden der Schnittstelle:
        - reserviere(Seminar se, Studierende st, Reservierungsdaten rd)
- Visualisierungsdienst
    - Klassen: Anmeldemaske, Suchmaske, Seminarübersicht, Reservierungs-Formular, Reservierungs-Bestätigung
    - Methoden der Schnittstelle:
        - Anmeldungsansicht()
        - Suchmaske()
        - Seminarübersicht()
        - ReservierungsFormular()
        - Reservierungsbestätigung()

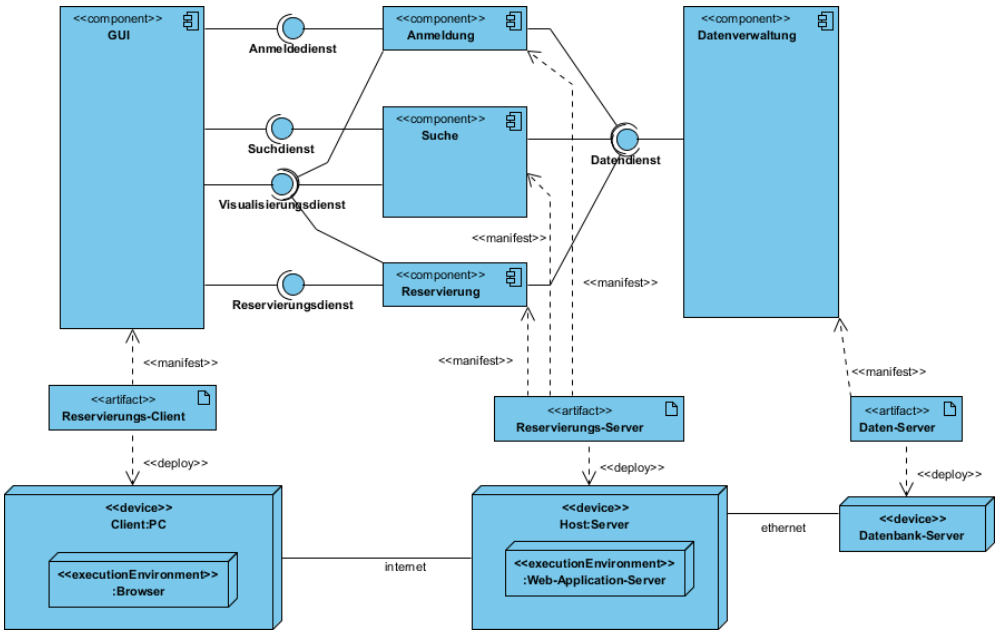
b) Identifizieren Sie Subsysteme. Erstellen Sie ein Komponenten-Diagramm (engl. „Component Diagram“) der Subsysteme. Notieren Sie ebenfalls die Dienste aus Aufgabenteil a) und zusätzlich deren Abhängigkeiten.



Aufgabe 2. *Deployment* (8 Punkte)

Zu Beginn des Projektes stellt Ihnen die auftraggebende Universität einen Server, eine Schnittstelle zum Internet sowie eine Studierenden-, Seminar- und Reservierungs-Datenbank zur Verfügung. Studierende sollen ihr System per Browser via Internet nutzen.

a) Zeichnen Sie ein Verteilungsdiagramm (engl. „Deployment-Diagram“), das die Hardware-Software-Zuordnung wiedergibt. **Hinweis**: Die Elemente von Verteilungsdiagrammen sind im Foliensatz zum Kapitel Systementwurf erläutert.



b) Die folgenden Entwurfsziele sind in unserer Anwendung zu beachten:

a. Nebenläufigkeit unabhängiger Workflows

b. Zentrale Verwaltung der Seminardaten

c. Schutz sensibler Daten der Studierenden

d. Zeitweise hohe gleichzeitige Zugriffszahlen

Welche der in der Vorlesung vorgestellten Architekturen machen in diesem Fall Sinn? Begründen Sie jeweils ihre Wahl.

- Model-View-Controller (MVC)
  - Model: Subsystem „Datenverwaltung“
  - View: Subsystem „GUI“
  - Controller: „Anmeldung“, „Suche“, „Reservierung“

- Repository Architektur:
  - Für viele unabhängige Nutzer, die nichts von einander wissen sollten
  - Zur Nutzung eines DBMSs
  - zentrale Lagerung des Domänenwissens vereinfacht den Umgang mit Nebenläufigkeit und Integrität.
  -
- Client-Server Architektur
  - Ist Spezialfall der Repository Architektur
  - Server entspricht dem Repository
  - Typisch: remote procedure call Mechanismus
  - Unterschied zu „normalem Repository“
  - Kontrollflüsse auf Client und Server bis auf Synchronisation für Anfragen unabhängig

Aufgabe 3. *Persistenz* (5 Punkte)

a) Welche Möglichkeiten des Datenmanagements kennen Sie aus der Vorlesung? Wie werden diese realisiert? Aufgrund welcher Anforderungen entscheiden Sie sich für die jeweilige Möglichkeit?

- Persistieren
  - Datenbank
    - Verschiedene Detailstufen der Daten für viele Nutzer
    - Heterogenes System (verschiedene Plattformen)
    - Verschiedene Anwendungen, die auf die Daten zugreifen
    - Komplexe Infrastruktur für das Datenmanagement
      - Sicherheit
      - Zuverlässigkeit
      - Hoher Durchsatz
  - (Temporäre) Dateien
    - Große Daten
    - „Raw“ Daten
    - Nur kurzzeitige Speicherung von Daten
    - Niedrige Informationsdichte
- Nicht Persistieren
  - Datenstruktur im Hauptspeicher

b) Welche Elemente von Aufgabe 1 müssen persistent sein, welche nicht? Warum?

- Immer wieder benötigte Elemente (Use-Case-Übergreifend):
  - Seminar
  - Reservierung
  - Studierende

⇨ **Persistieren**

- Nur temporär benötigt:
  - Controller und GUI-Elemente

⇨ **Nicht Persistieren**

Aufgabe 4. *Systementwurf - Architekturen* (6 Punkte)

Ältere Compiler wurden als **Pipe and Filter Architektur** realisiert, bei der jede Teilkomponente ihre Eingabe in eine weitere Zwischenrepräsentation transformierte, die als Eingabe der nächsten Komponente diente. Heutige integrierte Entwicklungssysteme verwenden hingegen eher eine **Repository-Architektur**.
***Hinweis**: Beachten Sie, dass mit Repository-Architektur im Systemdesign nicht CVS oder SVN gemeint ist!*

a) Diskutieren Sie die Entwurfsziele der beiden Architekturen, und schlussfolgern Sie, welche Ziele möglicherweise diese Änderung motiviert haben.

- Pipe and Filter Architektur
  - Einfaches Austauschen von Filtern: Flexibilität
  - Rekombination von Filtern (z.B. in Unix)
  - Möglichkeit der parallelen Verarbeitung
  - Hierarchien einfach zu erzeugen
  - Keine direkte Kommunikation unter Filtern
  - Datentransport zwischen den Filtern ggf. sehr aufwändig
    - Parallele Verarbeitung teilweise nur beschränkt möglich
    - falls Filter aufeinander warten
    - falls nur ein Prozessor existiert (Task-Wechsel-Overhead)
  - Wartbarkeit (Änderungen betreffen oft mehrere Filter)
  - Hierarchien schnell unübersichtlich -> hohe Komplexität
  - Hohe Kopplung der Subsysteme
- Repository Architektur
  - Eine einzige zentrale Datenstruktur. Die Subsysteme sind relativ unabhängig voneinander und ihr einziger Berührungspunkt ist das Repository
  - Subsysteme haben jeweils einen unabhängigen Kontrollfluss
  - Leichtere Handhabbarkeit von Nebenläufigkeit und Integrität zwischen Subsystemen
  - Neue Subsysteme leicht zu integrieren
  - Geeignet für Anwendungen mit komplexen, sich ändernden Datenverarbeitungs-Aufgaben.
  - Inkrementelle Updates
  - Kann Methoden und abhängige Teile genau dann übersetzen, wenn sie verändert wurden
  - Debugging und Fehleranzeige möglich, sobald Code geändert wurde
  - Änderungen am Repository haben starken Einfluss auf Subsysteme -> bottleneck

- Pipes and Filter Architekturen erlauben kaum inkrementelle Updates
- Repository Architekturen eignen sich gut für inkrementelle Updates

b) Geben Sie reale Beispiele für folgende Architekturen an. Begründen Sie dies und geben Sie wenn möglich auch passende Links dazu an.
- 3-Schichten
  - Res-On – Ergebnisse online Abrufen
  - https://reson.iai.uni-bonn.de/
- 4-Schichten
  - Webmail-Programm „SquirrelMail“
  - https://webmail.iai.uni-bonn.de/
- Client-Server
  - Elektronisches Vorlesungsverzeichnis „Basis“
  - https://basis.uni-bonn.de/