Übungen zur Vorlesung

Softwaretechnologie

- Wintersemester 2010/2011 -

Dr. Günter Kniesel

Übungsblatt 8 - Lösungshilfe

Aufgabe 1. *Anforderungs-Erhebung* (7 Punkte)

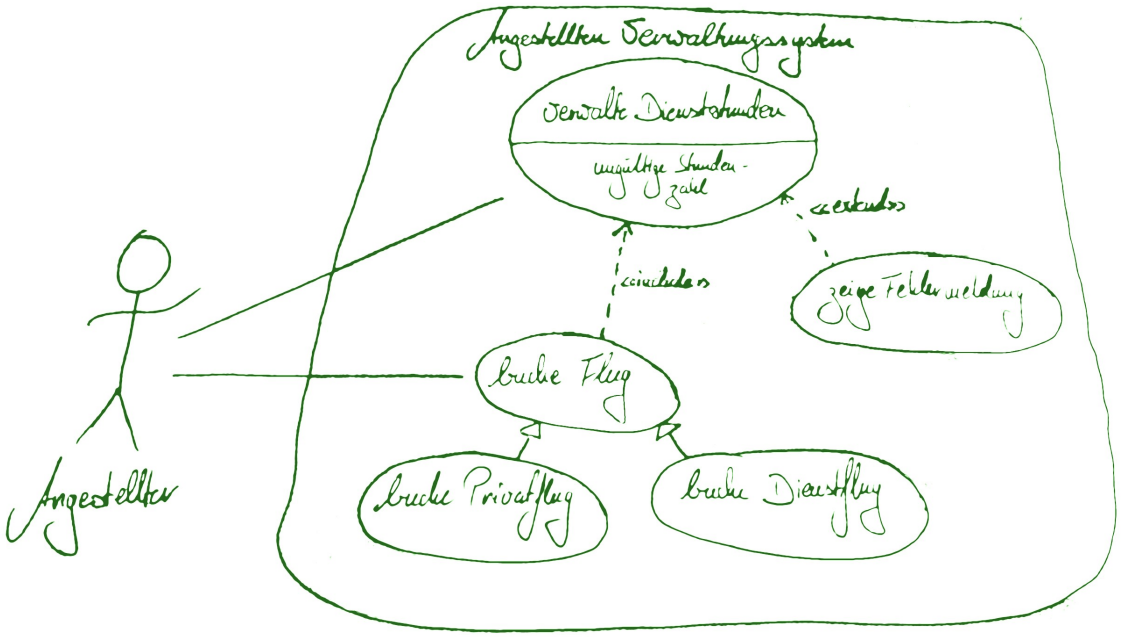
Hinweis: Bei dieser Aufgabe handelt es sich um eine ehemalige Klausuraufgabe. Versuchen Sie zur Übung daher zunächst, das Diagramm formal korrekt und vollständig auf Papier zu erstellen, bevor Sie die Lösung in Visual Paradigm nachzeichnen und einchecken.

Ziel dieser Aufgabe ist es, das interne Angestellten-Verwaltungs-System einer großen Firma funktional zu modellieren.

- Angestellte sollten in der Lage sein, ihre Flüge über das System zu buchen. Das System soll sowohl für dienstliche, als auch für private Flüge verwendet werden können.
- Des Weiteren sollen die Angestellten ihre Dienststunden mit dem System verwalten können. Flugstunden von dienstlichen Flügen sollen automatisch als Dienststunden angerechnet werden.

Die Eingabe von ungültigen Werten für die Dienststunden soll als Ausnahmefall betrachtet werden.

a) Zeichnen Sie ein Anwendungsfalldiagramm, um Ihre Lösung für das angegebene Problem darzustellen. Wenn Sie Abhängigkeiten zwischen Anwendungsfällen modellieren, begründen Sie diese.



- **<<extend>>:** Abhängiger Anwendungsfall modelliert Fehlerfalls -> Typisch für eine Extend-Beziehung
- **<<include>>:** Funktionalität zum Zugriff auf das Datenmodell wird normalerweise zwischen Anwendungsfällen geteilt
- **Generalisierung:** „Flug Buchen“ realisiert das allgemeine Verhalten. Die Anwendungsfälle „buche Privatflug“ und „buche Dienstflug“ enthalten spezielles Verhalten.

b) Welche nicht-funktionalen Anforderungen sind Ihrer Meinung nach für dieses System wichtig? Nennen Sie zwei passende nichtfunktionale Anforderungen und begründen Sie kurz Ihre Wahl.
Beispiele:
- Skalierbarkeit (Wenn die Zahl der Angestellten wächst)
- Performanz (Datenbankzugriffe sollten unter 10 Millisekunden liegen)
- Sicherheit (Dienststunden fallen unter das Datenschutzgesetz)

Aufgabe 2. *Analysephase* (12 Punkte)

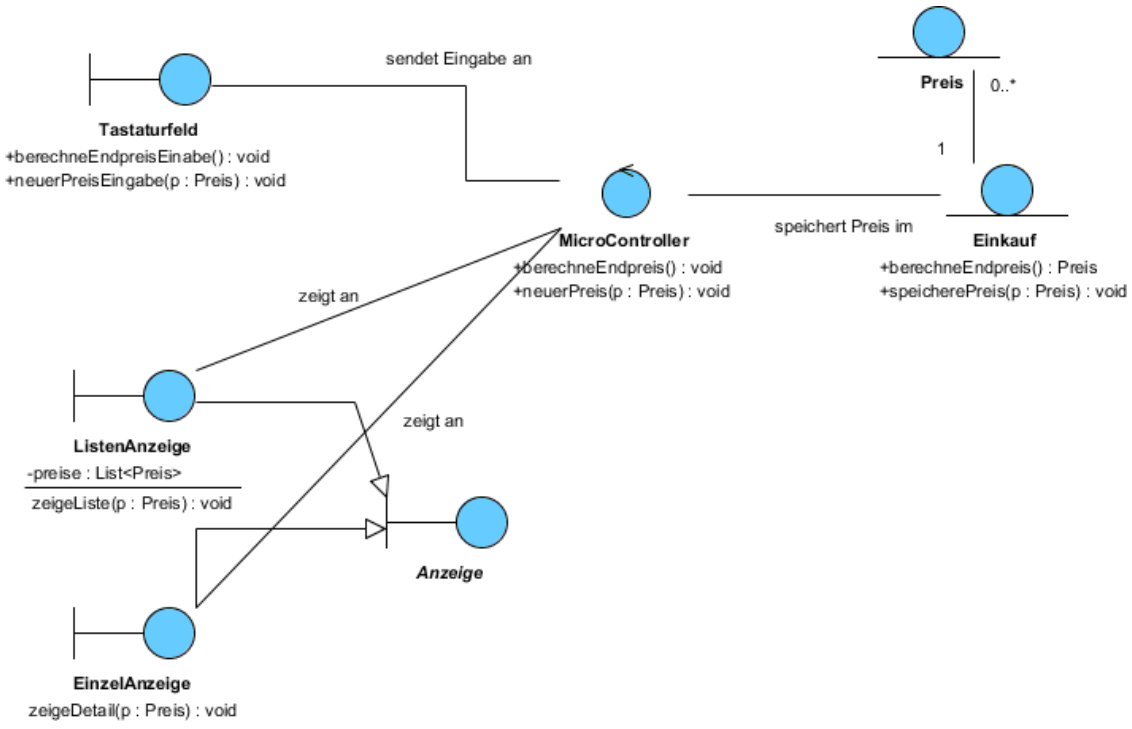
Eine Firma hat den Auftrag erhalten eine Kasse zu entwerfen. Die Kasse soll folgende Elemente enthalten:

- Ein Tastenfeld für die Eingabe von Preisen zu Artikeln und einer Taste zum Berechnen des Endpreises.
- Eine Listen-Anzeige für den Kassierer/die Kassiererin mit einer Auflistung der Preise des aktuellen Einkaufs.
- Eine Einzel-Anzeige auf Seite des Kunden, die den Preis des letzten eingegeben Artikels bzw. den Endpreis anzeigt.

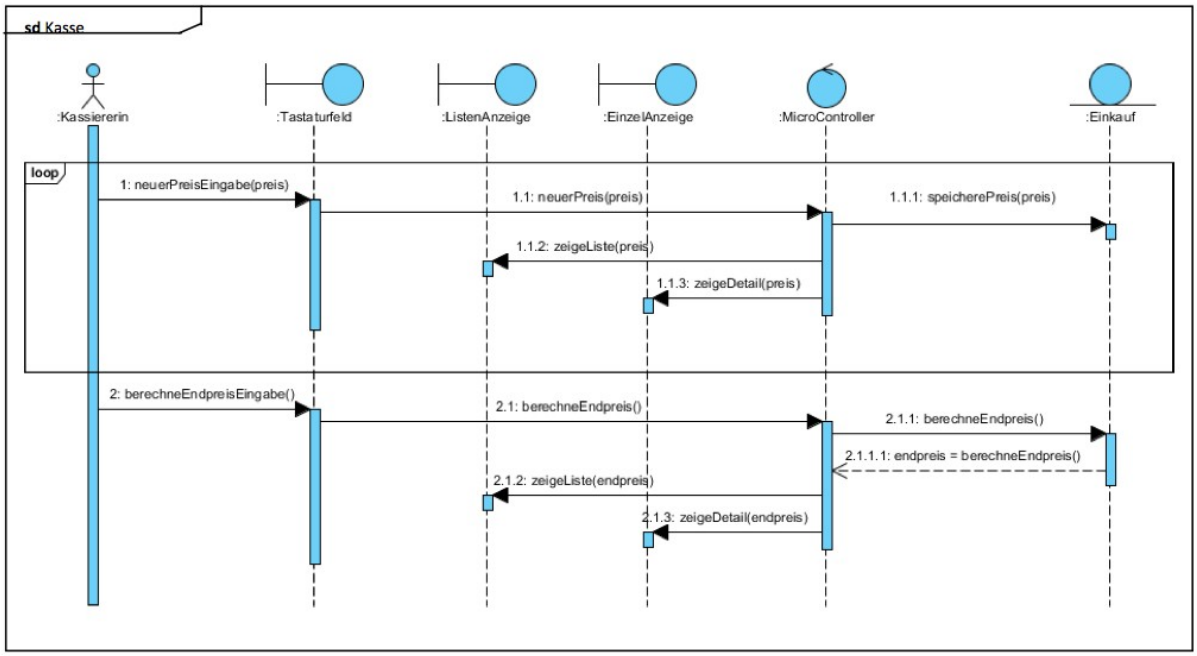
Die Durchführung eines Einkaufs wird durch folgende Sequenz von Ereignissen beschrieben:

- Der Kassierer/die Kassiererin tippt den Preis eines Artikels ein.
- Daraufhin werden alle registrierten Anzeigen aktualisiert.
- Die Preiseingabe kann beliebig oft wiederholt werden, bis der Kassierer/die Kassiererin die Taste zur Berechnung des Endpreises drückt, woraufhin der Endpreis angezeigt wird.

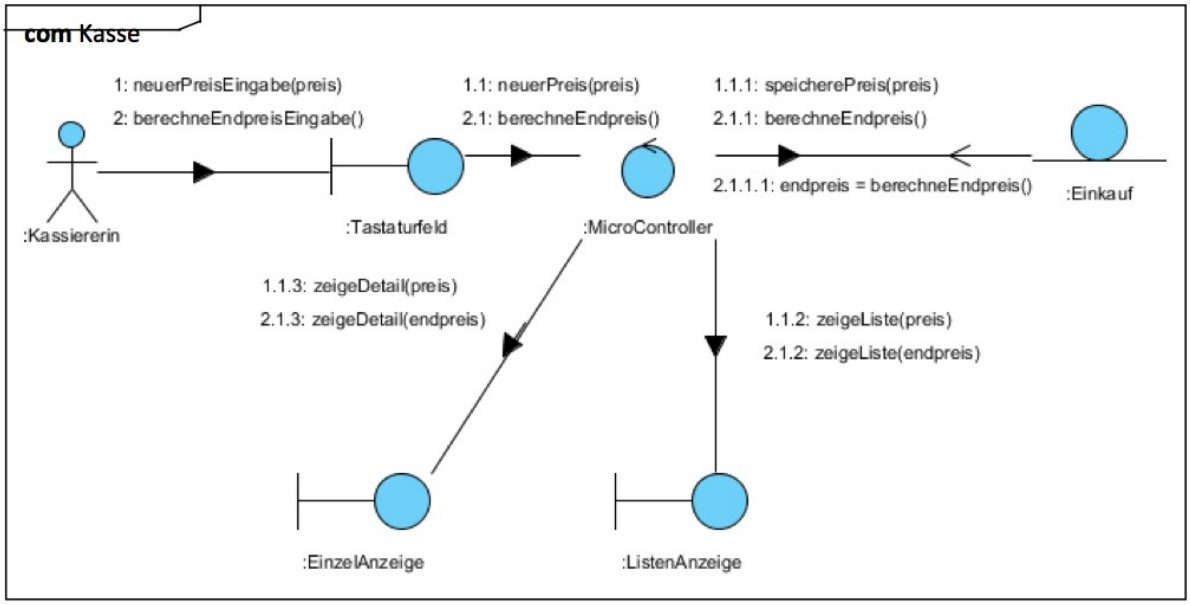
a) Zeichnen Sie ein **Klassendiagramm** für die Elemente der Kasse. Überlegen Sie sich Klassen, Operationen und Attribute. Fügen Sie Assoziationen ein, wo es Ihnen sinnvoll erscheint. Markieren Sie Boundaries, Controller und Entities.



b) Zeichnen Sie ein **Sequenzdiagramm** für obige Sequenz von Ereignissen.



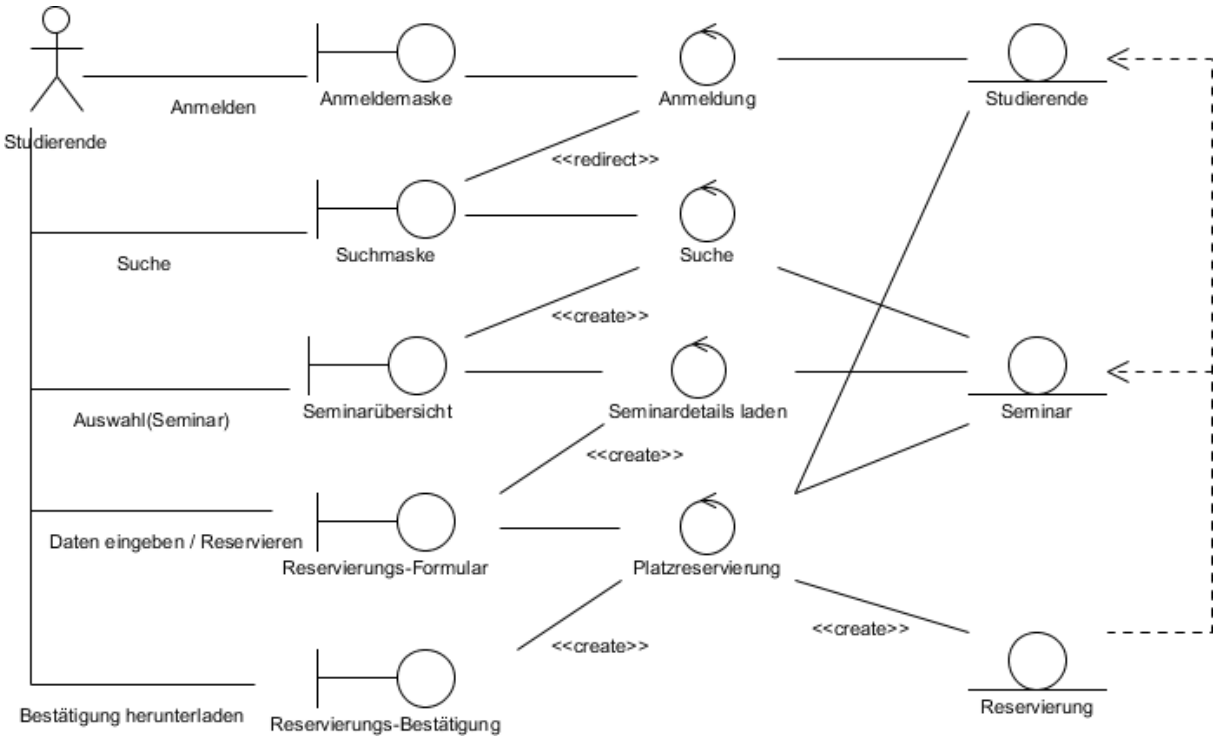
c) Zeichnen Sie ein **Kommunikationsdiagramm** für obige Sequenz von Ereignissen.



Tipp: Im Kontextmenü des Sequenzdiagramms finden Sie den Menüpunkt „Synchronize to Communication Diagram“

Aufgabe 3. *System-Entwurf* (11 Punkte)

Basierend auf nachfolgendem Analysemodell ist ein Seminar-Verwaltungssystem mit „Thin-Client“-Architektur zu entwerfen. Führen Sie, wie in der Vorlesung beschrieben, ein Systementwurf durch (Dienste Identifizieren, Aufteilung in Subsysteme, Anordnung der Subsysteme passend zur „Thin-Client“-Architektur. Entscheiden Sie sich im Zweifel für die einfachste Lösung, die funktioniert.

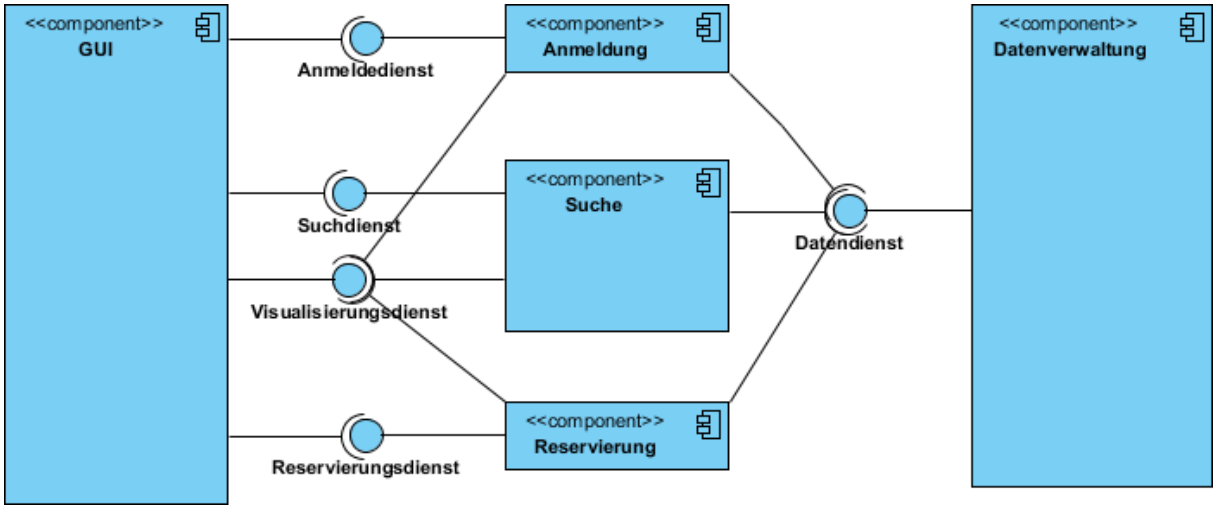


a) Definieren Sie sinnvolle *Dienste* (Interfaces mit komplett typisierten Operationen). Eine textuelle Aufzählung ist für diesen Aufgabenteil ausreichend (kein Diagramm erforderlich).

- 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 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

  Motivation für Architekturänderung: Inkrementelle Updates (z.B. kaum hat man etwas im Editor getippt, wird das getippte bereits geparst, analysiert, etc. und die Typanalyseergebnisse werden bereits auch schon von dem Editor genutzt, um eine „autocompletion“ vorzuschlagen).

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

b) Geben Sie mindestens ein reales Beispiel für jede der folgenden Architekturen an, wenn möglich mit passenden Links dazu. Begründen Sie Ihre Meinung.

- 2-Schichten-Client-Server
  - FTP-Client und -Server
- 3-Schichten-Client-Server
  - Übungsgruppen-Anmelde-System der SWT-Vorlesung
  - https://uebungen.iai.uni-bonn.de/
  - Browser, Webserver und Datenbank-Server für Veranstaltungen, Gruppen, Zuordnungen, …
- 4-Schichten-Client-Server
  - Webmail-Programm „SquirrelMail“
  - https://webmail.iai.uni-bonn.de/

  Oder:
  - Professionell erstellte Online-Shops haben typischerweise die Struktur: Browser, Webserver, Applikationsserver, Datenbankserver