# Probeklausur Architektur und Entwurfsmuster

# 2013/2014

## Ausgangssituation

Als Mitarbeiter eines Projektteams werden Sie mit der Entwicklung eines Kantinensystems betraut. Über dieses System sollen Kantinen, wie z.B. eine Hochschul-Mensa einfach abgebildet werden können.

## Produktvision:

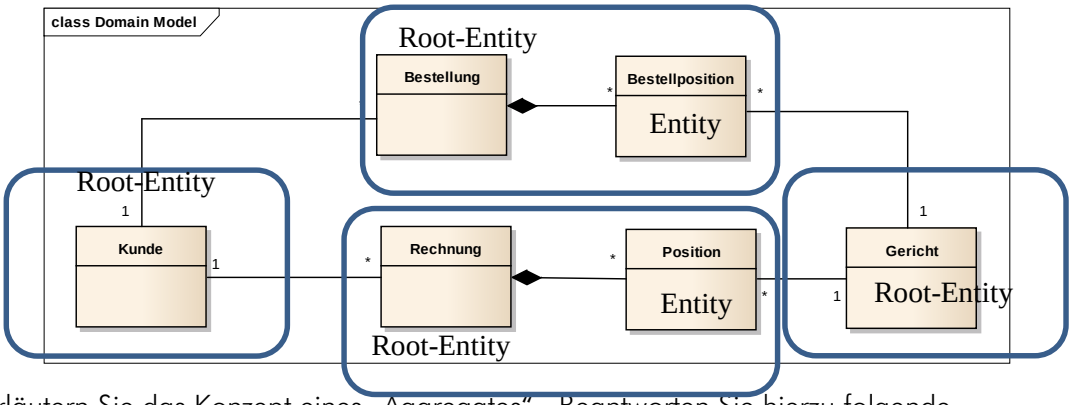
- Das System bietet die Möglichkeit den Speiseplan einer Woche online einzusehen. Dies soll sowohl per Webbrowser als auch über ein Smartphone möglich sein.
- Das System bietet dem Kantinenpersonal die Möglichkeit Speisepläne im Voraus zu erstellen und zu editieren.
- Das System verfügt über eine integrierte Lagerverwaltung. Diese kann automatisch Bestellvorschläge für die benötigten Zutaten auf Basis der Wochenpläne ermitteln.
- Zusätzlich bietet das System einen Bestell- und Lieferservice für externe Firmen und Privatkunden an. Firmen können die angebotenen Menüs vorab bestellen.
- Die Mitarbeiter der externen Firmen, bzw. die Privatkunden können auf Basis des Speiseplans ihre Menüwünsche online bestellen. Am jeweiligen Wochentag erfolgt die Auslieferung durch den Lieferservice der Kantine.
- Das System soll einen hohen Sicherheitsstandard erfüllen. Jeder Benutzer, der mit dem System arbeitet, unterliegt bestimmten Restriktionen, welche durch ein Berechtigungssystem definiert werden.
- Wichtig: Der Speiseplan kann von allen Benutzern eingesehen werden. Erst die Bestellfunktion erfordert eine Anmeldung am System.
- Die Wartung des Systems obliegt den Administratoren.
- Um eine hohe Akzeptanz zu erzielen, soll das System über eine Vielzahl von online-Schnittstellen verfügen.
- Sämtliche Zahlungsvorgänge sollen online über eine Schnittstelle zur Hausbank durchgeführt werden.

## Randbedingungen:

- Das System soll modular aufgebaut sein, so dass es jederzeit möglich ist, neue Module hinzuzufügen.

## 1. Software Architektur – Die Logische Sicht

1a) Im Rahmen des Designs wenden Sie nun die Methode „Domain Driven Design" auf das nachstehende Produktmodell an. Analysieren Sie das nachfolgende Diagramm und kennzeichnen Sie alle Klassen in diesem Diagramm. Stellen Sie folgende Kernelemente innerhalb des Diagramms heraus: Aggregate, Entity, Root-Entity, Datentypen. Verwenden Sie falls notwendig Stereotypen.




1b) Erläutern Sie das Konzept eines „Aggregates". Beantworten Sie hierzu folgende Aspekte: Was versteht man unter einem Aggregat? Was ist die sog. Root-Entity? Welche Einschränkungen existieren in Bezug auf aggregatübergreifende Assoziationen?

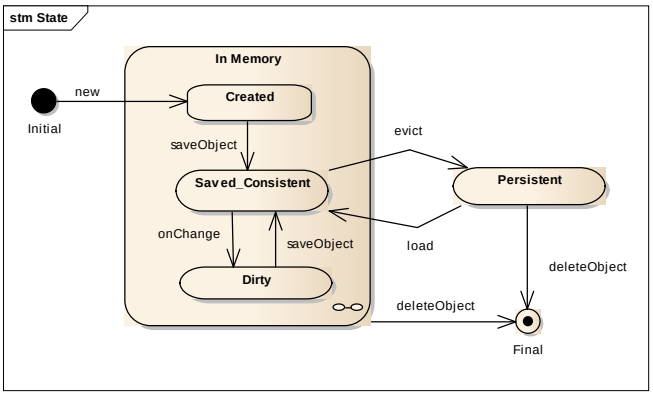
- Eine Gruppe von zusammenhängenden Objekten die als eine Einheit für den Datenaustausch verwendet werden.
- Das Aggregat kapselt alle Referenzen der beteiligten Objekte
- Jedes Aggregat hat einen ausgezeichnete Root-Entity.
- Außenstehende Objekte dürfen nur auf die Root-Entity verweisen.
- Alle Enitites (ausser der Root) haben eine locale Identität welche nur innerhalb des Aggregats eindeutig sein muss.
- Das Aggregat sorgt für die Einhaltung von Invarianten wenn immer sich ein Objekt innerhalb des Aggregats ändert.

- Eine Gruppe von zusammenhängenden Objekten die als eine Einheit für den Datenaustausch verwendet werden.
- Das Aggregat kapselt alle Referenzen der beteiligten Objekte
- Jedes Aggregat hat einen ausgezeichnete Root-Entity.
- Außenstehende Objekte dürfen nur auf die Root-Entity verweisen.
- Alle Enitites (ausser der Root) haben eine locale Identität welche nur innerhalb des Aggregats eindeutig sein muss.
- Das Aggregat sorgt für die Einhaltung von Invarianten wenn immer sich ein Objekt innerhalb des Aggregats ändert.


1c) Was versteht man unter sog. Intension Revealing Interfaces? Warum ist dieses Konzept wichtig, wie hilft es bei der Entwicklung?

- Benennen Sie Klassen und Operationen um deren Zweck bzw. Wirkung zu beschreiben, ohne Bezug auf die konkrete Realisierung zu nehmen.
- Dadurch wird die Wartbarkeit erhöht da das Verständnis der Software erleichtert wird.

1d) Erläutern Sie das nebenstehende Zustandsdiagramm. Dieses Zustandsdiagramm stellt den Lebenszyklus eines Domänen-Objekts dar. Erläutern Sie die Zustände sowie die zugeordneten Ereignisse. Stellen Sie den Unterschied zwischen einem Objekt einer Programmiersprache und einem Domänenobjekt heraus.




- Domänenobjekte existieren solange eine Repräsentation des Objektes im Hauptspeicher oder auf der Persistenz existiert
- Wir ein Objekt angelegt ist es nur im Speicher vorhanden. Durch das Speichern wird der Zustand des Objekts persistiert.
- Wird ein Objekt im Hauptspeicher verändert so ist der DB-Zustand inkonsistent. Erst ein erneutes speichern sorgt dafür dass die Zustände konsistent sind.
- Ein Programmiersprachenobjekt ist nur eine von vielen möglichen Repräsentationen.

1e) Erläutern Sie anhand des in Aufgabe 1d beschriebenen Lebenszyklus warum Assoziationen zwischen Aggregaten schwierig zu realisieren sind? Geben Sie eine Möglichkeit an, wie diese spezielle Form von Assoziation abgebildet werden kann.

- Aggregate haben unterschiedliche Lebenszeit bzw. Existenz im Hauptspeicher
- Deswegen ist die Implementierung schwierig wenn Objektreferenzen verwendet werden. Referenzen implizieren dass abhängige Aggregate gleichzeitig im Hauptspeicher sein müssen.
- Dies ist jedoch ein Wiederspruch.
- An dieser Stelle sollten Fremdschlüssel für die Realisierung verwendet werden.

## 2. Software Architektur

2a) Beschreiben Sie die 4+1 Sichten einer Architektur. Nennen Sie für jede Komponente die wichtigsten Eigenschaften/Ziele. Nennen Sie, falls möglich, jeweils eine Anforderung, welche durch eine Architektursicht gelöst wird.

- Struktursicht: Fokus: Beschreibt die statische Struktur der Software in Form von Subsystemen und Komponenten, Wiederverwendung
- Logische Sicht: Darstellung eines Implementierungsmodells. Ausgangspunkt ist ein Domänenmodell welches um Designaspekte und Komponenten erweitert wird, Funktionalität
- Physikalische Sicht: Zuordnung der Software auf die physische Hardware sowie Verteilung ( = Distributed System ), Availability
- Ablauf Sicht: Abbildung des Produktmodells auf ein Verarbeitungsmodell. Behandlung von Nebenläufigkeit und Synchronisation, Skalierbarkeit
- Szenarien: Entwicklung & Qualitätssicherung der erstellten Architektur

2b) Was versteht man unter dem Architekturprinzip „Information Hiding“? Was passiert, wenn man gegen das Architekturprinzip verstößt?

- Die Art und Weise wie ein Systemteil seine Aufgabe erfüllt muss im inneren des Moduls verborgen bleiben
- Über das Modul sind nach außen nur Informationen bekannt welche über eine Schnittstelle explizit zur Verfügung gestellt worden sind
- Über eine Schnittstelle wird so wenig wie möglich nach außen hin preis gegeben
- Das System wird instabil bei Änderungen

## 2. Software Architektur (Fortsetzung)

2c) Welche Gründe sprechen für die Zerlegung einer Anwendung in Komponenten?

- Durch die Zerlegung in Komponenten wird die einhaltung der folgenden Architekturprinzipien gefördert
  - Seperation of Concerns
  - Lose Kopplung
  - Wiederverwendbarkeit
- Darüber hinaus stellen Komponenten eine logische Strukturierung dar welche zur Fortschrittskontrolle, Arbeitsplanung, Aufwandsplanung usw. verwendet werden können.

2d) Welches Merkmal einer Komponente gibt Aufschluss darüber, wie gut eine Komponente wiederverwendet werden kann? Begründen Sie Ihre Aussage. Wann ist eine Komponente sehr gut, bzw. schlecht wiederverwendbar?

- Anzahl der importierten Schnittstelle
- Je weniger Schnittstellen importiert werden umso niedriger ist die Abhängigkeit zu der Umgebung
- Wenn keine Abhängigkeiten existieren ist die Komponente sehr gut wiederzuverwenden.

## 3.) Software Architektur – Die Struktursicht

Im Rahmen einer Architekturbewertung sollen Sie nun Komponenten, sowie deren Schnittstellen bewerten. Hierzu wird ihnen für die Persistenzkomponente des Kantinensystems folgendes Quellcodefragment vorgelegt (Schnittstellenspezifikation):

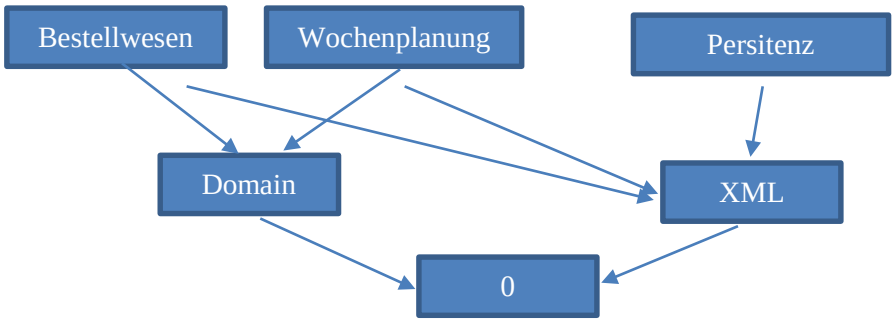
```
public interface PersistenceKantine {

    // Gericht aus Datenbank laden
    XMLDocument loadGericht( String gerichtID  );

    // Gericht in der Datenbank speichern
    void saveGericht( XMLDocument docGericht );
}
```

3a) Skizzieren Sie den Software-Kategoriegraphen unter der Annahme, dass die Anwendungskomponente „Wochenplanung“ sowie die Anwendungskomponente „Bestellwesen“ auf die Persistenzkomponente zugreifen. Darüber hinaus gelten folgende Annahmen:

- Alle XML spezifischen Klassen/Interfaces liegen in der Kategorie „XML“
- Die Klassen des Produktmodells werden in der Kategorie „A“ (Anwendung) angesiedelt

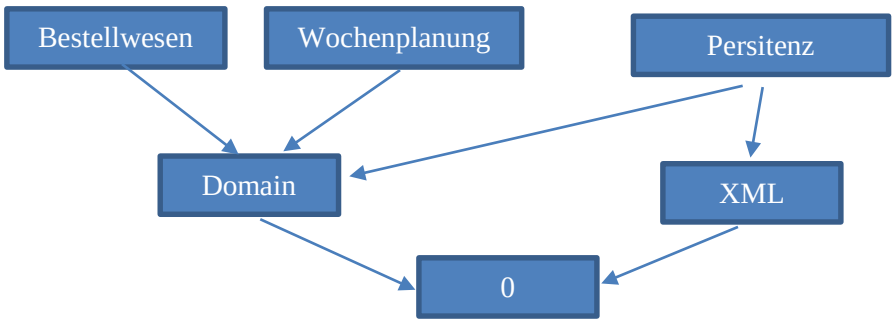



3b) Nennen Sie zwei Probleme, die in dieser Architektur enthalten sind. Wie hilft R-Software bei der Lösung des Problems und welche Einschränkung gilt bei der Verwendung von R-Software?

- Bestellwesen und Wochenplanung sind abhängig von der Technologie XML.
- Dadurch wird das Verständnis erschwert
- Ein Austausch der Persistenztechnologie ist nicht einfach möglich
- R-Software: Transformationssoftware welche ausschließlich zwischen Domänenwelt und Technologie übersetzt. R-Software realisiert keine Businesslogik

## 3. Software Architektur (Fortsetzung)

3c) Zeichnen Sie nun einen verbesserten Kategoriegraphen. Begründen Sie warum Ihr Vorschlag die genannten Probleme löst.




- Bestellwesen und Wochenplanung sind nun reine Kategorie
- Persistenz wird als R-Software realisiert.

3d) Beschreiben Sie die Sichtbarkeitsregeln, welche Quasar basierend auf einem Kategoriengraphen definiert. Gehen Sie insbesondere darauf ein, welche Auswirkung auf die Schnittstellen sowie Parameter existiert. Welche besondere Stellung hat die Kategorie-0?

- Eine Komponente darf nur Artefakte sehen welche in der gleichen Kategorie liegen
- Eine Komponente darf Artefakte sehen welche direkt oder indirekt in einer Vorgängerkategorie liegen
- Auch eine Schnittstelle ist einer Kategorie zugeordnet. Schnittstellen sind so zu definieren dass nur Parameter verwendet werden welche gemäß Sichtbarkeitsregeln erlaubt sind
- In der Kategorie 0 ist die allgemeine Weisheit der Informatik enthalten. Also dass Wissen dass für alle zugänglich ist.

3e) Definieren Sie den Begriff „unreine Softwarekategorie“. Warum sollte man diese Softwarekategorie vermeiden, welches Problem existiert?

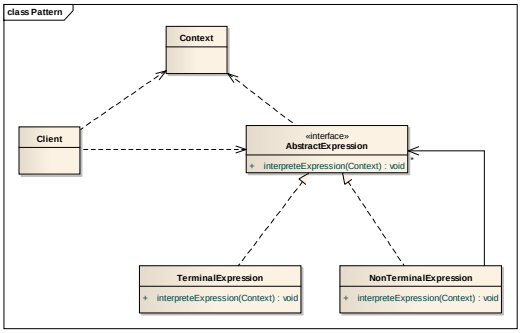
- Eine unreine Softwarekategorie hat mehr als einen Vorgänger im Kategoriegraphen
- Es ist mehr Wissen notwendig um Komponenten dieser Kategorie zu verstehen

## 3.) Software Design – Pattern

3a) Erläutern Sie die Grundidee/-problem welches durch den Einsatz von Erzeugermuster gelöst werden soll

- Erzeugermuster löschen das Problem der Objekterzeugung
- Wird ein Objekt erzeugt so entsteht eine hohe Abhängigkeit zur Implementierungklasse des Objektes
- Diese Abhängigkeit kann schlecht bezogen auf die Flexibilität eines Programmsystems sein.

3b) Erläutern das nachfolgend dargestellte Interpreter-Pattern. Wann ist der Einsatz dieses Patterns sinnvoll und wo liegen die Grenzen.




- Interpretiere die Sätze einer Grammatik dynamisch
- Context: Enthält Information für die Berechnung welche zwischen den Ausdrücken übergeben werden kann
- AbstractExpression: Abstrakte Sicht auf einen Ausdruck. Der Ausdruck berechnet ein Ergebnis. Das wie ist den konkreten Implementierungen überlassen.
- TerminalExpression: Ein einfacher Ausdruck welcher durch sich selbst beschrieben ist und keine weiteren Unterausdrücke benötigt.
- NonTerminalExpression: Ein zusammengesetzter Ausdruck dessen Ergebnis sich aus den Teilergebnissen der Unterausdrücke berechnet.

Grenzen: Interpretierte Programme sind in der Regel langsamer → Performance-Engpass

3c) Skizzieren Sie ein Klassendiagramm welche das Interpretermuster auf die Berechnung von einfachen arithmetischen Ausdrücken wie z.B. 5 * ( 4 – 2 ) anwendet.