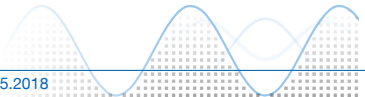
Grundkurs C++

Objektmodellierung

Martin Knopp, Martin Gottwald, Stefan Röhrl

09.05.2018



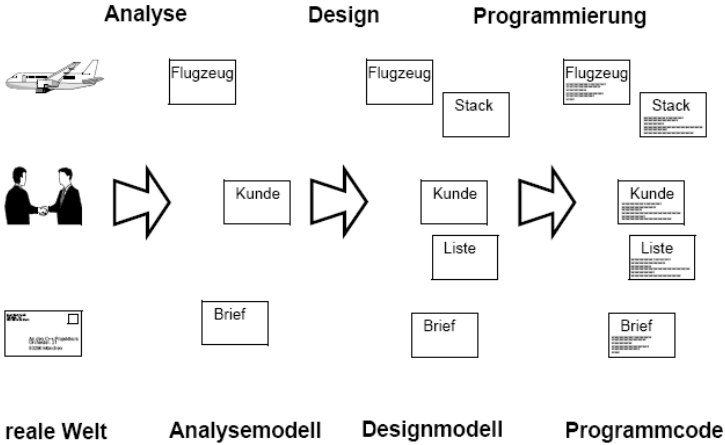
Objektmodellierung

- Welche Objekte bzw. Klassen werden benötigt?
- Welche Information wird benötigt, um ein Objekt zu beschreiben?
- Welche Beziehungen bestehen zwischen Klassen?
- Welche Beziehungen bestehen zwischen einzelnen Objekten?

Es wird eine methodische Vorgehensweise benötigt, um diese Fragen zu beantworten!
Es gibt viele Methoden, hier nur eine Übersicht über einzelne Techniken.

Die Zwischenergebnisse sind in einer sinnvollen Notation festzuhalten.
Die Notation, die am weitesten verbreitet ist: UML

Objektmodellierung: Arbeitsschritte



Zielsetzung und Vorgehen

- Struktur des Problembereichs soll möglichst genau auf die Implementierung abgebildet werden.
- Sanfter Übergang zwischen des Phasen des objektorientierten Entwurfsprozesses.
- Durchgängigkeit der Modelle von Analyse über Design bis zur Implementierung.
- Stabilität des Entwurfsprozesses erleichtert Wiederverwendbarkeit von Analyse-, Design- und Programmierergebnissen.
- Ein gutes Design erfordert den Aufbau entsprechender Erfahrung mit abzubildendem Problembereich und dessen Abläufen.

Modelltypen

- Beschreibung eines Systems nicht durch ein einziges Modell
- Alle Methoden verwenden daher mehrere Modelle

$\Rightarrow$ Aufdecken von Unvollständigkeiten und Widersprüchen

Unterscheidung zwischen:

- statischen und dynamischen Modellen
- logischen und physikalischen Modellen

Logisch/physikalisch und statisch/dynamisch sind orthogonale Unterscheidungen

Objektorientierte Analyse (OOA)

- Untersuchung und Beschreibung des Systembereichs
- Unabhängig von der Programmiersprache
- Präzisierung des Problems
- Wichtig: Klärung der Frage, „was“ das System tun muss, nicht, „wie“ es zu implementieren ist

Nutzen und Ziele der OOA

- Verringerung der Komplexität: Herausfiltern der für die Aufgabe wichtigen Aspekte
- Visualisierung:
  - Diagramme machen Ideen klarer: Es wird überprüfbar, ob sie praktikabel sind
  - Erkennen von vergessenen Schnittstellen, unnötigen Komponenten oder konzeptionellen Schwächen
- Kommunikation mit Problemexperten, Anwendern oder Kunden über einfache aber eindeutige Notation
- Wiederverwendbarkeit von Analyseergebnissen
- Testen von Einheiten bereits vor ihrer Fertigstellung (Beispiel: Durchspielen von Szenarien in Simulationen)
- Aus der OOA kann ein Prototyp abgeleitet werden

Objektorientiertes Design (OOD)

- Abbilden des Realitätsmodells auf den Lösungsbereich
- Vorstufe der Umsetzung in eine Programmiersprache
- Hinzufügen von Klassen, Attributen und Methoden, die für die Realisierung gebraucht werden (z. B. Datenhaltung in Form einer verketteten Liste)
- Einbeziehung bestehender Entwürfe und Klassenbibliotheken (z. B. UI-Library) in das Design
- Designüberlegungen, wie Wahl des Betriebssystems und der Programmiersprache
- Übergang von OOA nach OOD fließend, Zielsetzung ist jedoch sehr unterschiedlich

Objektorientierte Programmierung (OOP)

- Umsetzung des erstellten Designmodells in eine Implementierungssprache
- Basis ist das Ergebnis der OOD, nicht mehr das OOA
- Iterative Rückkehr zu OOA und OOD möglich

Software-Lebenszyklus

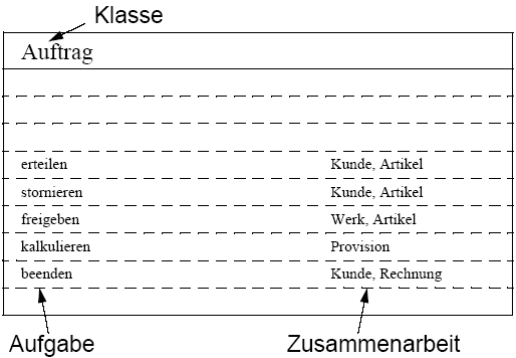
- Gesamter Software-Lebenszyklus ist integraler Bestandteil der objektorientierten Methoden
- Integration von Tests in OOA, OOD und OOP
- Während der Wartungsphase kann in die Analyse- und Designphase zurückgegangen werden
- Erleichterte Wiederverwendbarkeit von Ergebnissen der Analyse-, Design- und/oder Programmierphase

Schritte der OOA

- Ausgangspunkt: Problembeschreibung in Form eines Pflichtenhefts o. ä.
- Identifizierung von Klassen und Objekten
- Identifizierung der Beziehungen zwischen den Klassen und Objekten
- Beschreibung der Klassen und Objekte durch Definition von Attributen und Methoden
- Bildung von Subsystemen und Definition logischer Abläufe
- Zielsetzung: Nachbildung der Realität

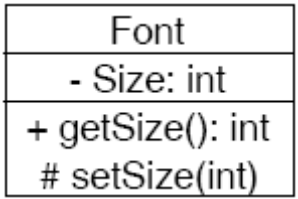
CRC-Methode

- einfache, aber wirkungsvolle Methode
- eine Karteikarte (CRC-Karte) für jede Klasse (Class): Eintrag von Aufgaben (Responsibilities) und Beziehungen zu anderen Klassen (Collaborations)

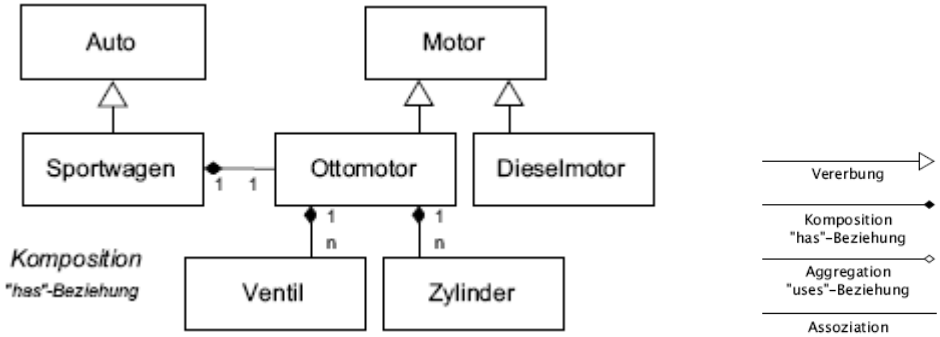


Klassendiagramm

- statische / logische Sicht
- Ein oder mehrere Klassendiagramme beschreiben die Klassen des Systems und ihre Beziehungen untereinander
- Unterschiedlicher Typ von Klassen: normale Klasse, parametrierte Klasse, instanziierte Klasse, abstrakte Klasse
- *wichtige* Methoden und Attribute sollen im Klassendiagramm dargestellt werden



Klassendiagramm: Assoziationen und Vererbung

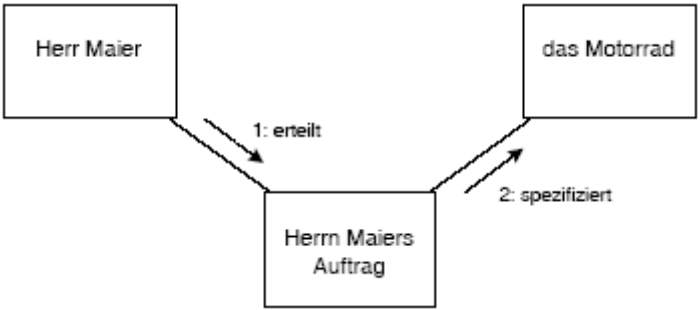


Zustandsdiagramm

- Zustandsdiagramme können als Ergänzung der Klassenbeschreibung verwendet werden
- Einfache oder hierarchische (nebenläufige) Zustandsdiagramme zur Beschreibung der möglichen Zustandsübergänge innerhalb einer Klasse

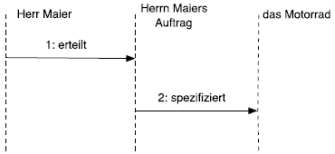
Objektdiagramm

- dynamische / logische Sicht
- Momentaufnahme des Systems (da Objekte flüchtig sein können)
- Beziehungen aus Klassendiagrammen spiegeln sich in Objektdiagrammen wider



Interaktionsdiagramm

- dynamische / logische Sicht
- Ergänzung der Objektbeschreibung (bessere Visualisierung, keine andere Information)
- Zeitablaufdiagramme mit Objekten (vertikale Linien) und Nachrichten (horizontale Pfeile)
- keine Übergabeparameter, Return-Werte
- Modellierung von Nebenläufigkeit durch Markieren von gleichzeitig aktiven Objekten möglich
- In UML: Kollaborationsdiagramme, Sequenzdiagramme



Identifizierung von Klassen und Objekten

- Analyse: Suchen von Klassen und Objekten im Problembereich
- Design: Identifizierung von Klassen, die zur Abbildung auf Software nötig sind (z.B. verkettete Listen)
- Implementierung: Identifizieren von weiteren Klassen, die die Systemarchitektur vereinfachen
- Vorgehen:
  - Heuristiken
  - Szenarien zur Beschreibung von Ereignissen
  - „Use-Case-Analysis“ zur Betrachtung von Systemabläufen
  - Systementwickler muss sich mit der Terminologie des Systembereichs auseinandersetzen
- Unterstützung durch Techniken wie CRC-Karten, Data Dictionaries (in die alle Identifizierten Klassen und ihre Eigenschaften eingetragen werden)

Identifizierung der Semantik der Klassen und Objekte

- Betrachten des Verhaltens der Klassen und Objekte (nicht der Attribute)
- Unterstützung durch CRC-Karten, später durch Diagramme zur Darstellung von Abläufen (z.B. Sequenzdiagramme)
- sowohl isolierte Betrachtung der Klasse als auch Suche nach Mustern und Gemeinsamkeiten zwischen Klassen
- schrittweise Verfeinerung
- Festlegung des public-Teils des Klassenheaders

Identifizierung der Beziehungen zwischen den Klassen

- besonders kreativer Teil des Entwurfs
- Erkennung kooperierender Objekte und Systemgrenzen
- Beschreibung von Beziehungen wie Vererbung, Assoziation und (später) Aggregation oder Instanziierung von generischen Klassen
- Verfeinerung der Klassen- und Objektdiagramme
- Neugestaltung der Klassenorganisation (z. B. durch Einführung von Vererbungsbeziehungen)

Spezifikation der Implementierung

- Festlegung des inneren Aufbaus von Klassen (Attribute und Methoden)
- Identifikation weiterer Klassen
- Ziel: detaillierte Spezifikationen, Klassen- und Zustandsdiagramme
- Modellierung der physikalischen Architektur

Literatur

Literatur:
Balzert, Heide: *Lehrbuch der Objektmodellierung*
Zuser, Wolfgang: *Software Engineering mit UML und dem Unified Process*

Links:
http://www.torsten-horn.de/techdocs/uml.htm

Hausaufgabe

- Präsentation Analyse- und Designmodell fürs Hauptprojekt am 16.05. (Team 1–4) und 23.05. (Team 5–7)
- Spielidee detailliert ausarbeiten unter Berücksichtigung der Anforderungen
- Analysemodell muss alle Komponenten der Spielidee umfassen
- Designmodell bildet das Analysemodell exakt ab

Anforderungen / Hauptprojekt

- Entwicklung eines Physikspiels
- Schwerkraft als Eigenschaft reicht nicht aus (also kein Jump&Run-Spiel)
- Mehrere Level
- Unterschiedliche Spielelemente
- Spielelemente sollen über physikalische oder technische Eigenschaft verknüpft werden
- Das Spiel muss unter Linux kompilierbar sein, soll unabhängig vom Speicherort laufen (keine festen Pfade)
- Sinnvolle Dokumentation des Projekts bzw. des Quellcodes (Doxygen könnte ein geeignetes Tool sein)

Optionale Anforderungen (zwei, vgl. Homepage!)

- Leveleditor
- Soundausgabe
- Highscore und Statistik
- Speichern & Laden
- Dynamische Grafikelemente und ausgefeilte Animationen
- Mehrspielermodus
- dito mit Netzwerkunterstützung
- …