Vorlesung
Software aus Komponenten

2. Grundlagen

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2006/07

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Was sind Komponenten und was nicht?

- „Component Software – beyond object oriented programming“
- Problem der Abgrenzung der Begriffe „Objekt“ und „Komponente“
  - synonym in manchen (Kon)texten
  - ex. auch Begriffe wie „Komponenten-Objekt“
- Objektbegriff: Instanz einer Klasse (Java) oder Clon eines Prototyp-Objekts (Factory-Pattern)
- Gemeinsamkeiten:
  - Dienste werden über Schnittstellen angesprochen
  - Schnittstellen nach Beschreibungsregeln typisiert und kategorisiert
  - Beschreibung nach Mustern (Syntax) und innerhalb von Frameworks (Semantik)
- weitere Verwirrung durch Sprachdesigner:
  - Namensräume, Moduln, Packages

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Notwendigkeit, in diesem Universum von Namen und Konzepten **aufzuräumen**, Begriffe an definierte **Bedeutungen** zu binden und in ein gewisses **Ordnungsschema** zu bringen.

- Ansatz: Bedeutungen von Begriffen durch Angabe charakteristischer Eigenschaften fixieren

Komponentendefinition

Charakteristische Eigenschaften einer Komponente:

1. Einheit unabhängiger Packung
2. Einheit der Komposition durch Dritte
3. ohne (extern beobachtbaren) Zustand

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Folgerungen aus der Definition

- Einheit unabhängiger Verteilbarkeit
  - separiert von Umgebung und anderen Komponenten
  - Kapselung charakteristischer Merkmale
- Einheit unabhängiger Verteilbarkeit
  - nie in Teilen zu verteilen
- Komposition durch Dritte
  - Verwendung ohne Kenntnis konstruktiver Details
- Komposition durch Dritte
  - genügend selbstabgeschlossen
  - klare Spezifikation der Anforderungen und Angebote

- zustandslos
  - ohne diese Forderung hätten keine zwei Installationen „derselben“ Komponente garantiert gleiche Eigenschaften
  - Zustand spielt sich ausschließlich auf der Ebene der Objekte ab

- Zustandslos
  - Kann nicht von Kopien unterschieden werden
    - Ausnahme: Zustandsparameter jenseits von Funktionalität
  - Intern verwendete Zustände (oft aus Gründen der Leistungsfähigkeit) dürfen von außen nicht sichtbar sein
    - Beispiel: Cache
  - Nicht sinnvoll, in einer Umgebung mehrere Instanzen einer Komponente zu halten
    - Achtung Konfusion vermeiden! Beispiel Datenbank. Unterscheide zwischen Datenbank-Server-Programm (Komponente) und darauf laufender Datenbank als „Instanz“ des Datenbank-Konzepts.
    - Diese Unterscheidung zwischen dem (fest verdrahteten) „Plan“ (= Komponente) und den (sich im Laufe der Zeit ändernden) „Instanzen“
    - Analog der Unterscheidung zwischen Objekten und Objektzuständen

- Komponenten sind heute praktisch meist schwergewichtige Einheiten mit genau einer Instanz pro System

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Objektdefinition

Charakteristische Eigenschaften eines Objekts:

1. Einheit der Instanziierung mit (eindeutiger) Identität
2. kann (extern beobachtbaren) Zustand besitzen
3. kapselt Zustand und Verhalten

Folgerungen:

- Einheit der Instanziierung
  - partielle Instanziierung nicht möglich

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- eindeutige Identität
  - Zustand für jedes Objekt individuell, kann sich im Laufe des Objekt-Lebenszyklus ändern
  - einzig garantiert persistente Eigenschaft eines Objekts ist dessen abstrakte Identität
    - Werner Brösels Auto bleibt W.B. Auto, auch wenn W.B. im Laufe der Zeit alle Teile daran ausgetauscht hat (und das Auto von seinem Original nichts mehr hat als eben diese abstrakte Identität)

- Einheit der Instanziierung
  - Plan muss Zustandsraum, Anfangszustand und (anfängliches) Verhalten eines neuen Objekts beschreiben
  - Plan muss vor der Existenz der Objekte existieren
  - Plan kann explizit sein = Klasse
  - Plan kann implizit sein = Prototyp-Objekt
  - Initialzustand muss valide sein, kann aber von weiteren Parametern abhängen
    - Code zur Erzeugung kann statisch sein = Konstruktor
    - Code zur Erzeugung kann selbst Objekt sein = factory object
    - Objekte können von anderen Objekten erzeugt werden = factory method

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten und Objekte

- Komponenten entfalten ihre Wirkung in der Regel über Objekte
  - Eine Komponente besteht also meist aus mehreren Klassen oder unveränderlichen Prototyp-Objekten sowie weiteren Komponenten-Ressourcen
- Komponenten können auch vollkommen anders realisiert sein
- Export von Objekt(referenzen) bedeutet nicht, dass es intern auch OO zugeht und kann (allein über Objektreferenzen) von außen auch nicht erforscht werden
- Komponenten und Klassen
  - Komponenten können mehrere Klassen umfassen
  - eine Klasse kann nicht über mehrere Komponenten verteilt sein

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Komponenten und Vererbung
  - Vererbung ist über Komponentengrenzen hinweg möglich
    - muss aber in der Komponenten-Schnittstelle über eine Import-Deklaration explizit verankert sein
  - Vererbung von Spezifikationen (Schnittstellen-Vererbung)
    - wesentliche Technik, um Korrektheit über Komponentengrenzen hinweg zu garantieren
    - gängiger Weg zur Operationalisierung von Standards
  - Vererbung von Implementierungen
    - relativ schwierige Problematik über Komponentengrenzen hinweg
    - Thema greifen wir später noch mal auf

Moduln

- Komponenten sind eher Moduln ähnlich. Wo sind die Unterschiede?
- Typische Eigenschaften von Moduln:
  - separate Compilierbarkeit
  - Mechanismen zur Typprüfung über Modulgrenzen hinweg
- Behauptung [Meyer 1988]: Klassen sind das bessere Modulkonzept
  - Sicht auf Moduln als ADT
  - Klasse = ADT + Polymorphie + Vererbung
  - Aber: Moduln unterstützen keine Instanziierung
    - statische Klassen als Modul-Imitation im Klassenkontext
- Moderne Sprachen (Modula-3, Oberon, C#) trennen das wieder
  - Moduln können mehrere Klassen umfassen
    - C#: Assemblies
    - Java: Imitation durch innere Klassen

- Im Gegensatz zu Klassen können Moduln die Grundlage für minimale Komponenten bilden
  - Beispiel: math-Bibliotheken
    - sind eher funktionaler als objekt-orientierter Natur
- Moduln unterstützen keine Zuordnung persistenter unveränderlicher Ressourcen (jenseits hart im Code „verdrahteter“ Konstanten)
  - Komponenten werden mit lokal verfügbaren Ressourcen zu einer „lokalen Komponente“ konfiguriert
    - die Beziehung zwischen einer Komponente und ihren Lokalisierungen ist komponenten-technologisch wichtig, im Modulkontext ausgeblendet
- Nicht jeder Modul geht als Komponente durch
  - erlaubt: globale Variable, statische Abhängigkeit von Implementierungen in anderen Moduln (= zustandsbehaftet)
- Zusammenfassung: Modularität ist eine Voraussetzung für Komponenten-Technologie
    - Bindung im Modul so hoch wie möglich, Kopplung zwischen Moduln so gering wie möglich [Parnas 1972]
    - selbst das ist heute noch längst nicht Standard

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Whitebox und Blackbox

- Thema: Sichtbarkeit der Implementierung hinter der Schnittstelle
- Blackbox: Der Nutzer weiß nichts über die Interna
  - Konzept der Komponente als Menge von Schnittstellen und deren Spezifikation
    - Bsp: API-Programmierung
- Whitebox: Die Schnittstelle kontrolliert die Zugriffsrechte, Interna sind aber prinzipiell bekannt
  - Konzept der Komponente als Software-Fragment
    - schwache Form: durch Studium zusätzliche Informationen über das interne Verhalten der Komponente bekommen (glassbox)
    - starke Form: Implementations-Vererbung
  - Entwickler studieren die Implementierung
    - Problem des Release-Wechsels
  - Viele Klassenbibliotheken und Frameworks fallen unter diese Kategorie

Definition Software-Komponente

Eine Software-Komponente ist eine Einheit der Komposition mit durch Kontrakt spezifizierten Schnittstellen und nur expliziten Kontext-Abhängigkeiten. Eine Software-Komponente kann unabhängig verteilt werden und zur Komposition durch Dritte verwendet werden.

- Definition wurde erstmals so 1996 auf der „European Conf. on OO Programming“ gegeben [Szyperski, Pfister]
- technische Seite: Unabhängigkeit, Schnittstellen-Kontrakt, Zusammenbau
- soziale Seite: Dritte, Verteilung
- Diese Verbindung ist typisch für den Komponentenbegriff nicht nur im Software-Bereich

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Schnittstellen-Kontrakt

- Muss die Verwendung der Komponente in einem Produktiv-System genau beschreiben
  - Aspekte:
    - Schnittstellen im engeren Sinne
    - Entpackung, Konfiguration, Installation der Komponente
    - Instanziierung und Beschreibung des Verhaltens dieser Instanzen durch ihre Schnittstellen (Laufzeitverhalten)
    - Beschreibung kollektiver Phänomene
      - Beispiel: Eine Stapel-Komponente hat Methoden `push` und `pop` und es muss klar sein, dass `pop(push(o))` wieder o zurückgibt
      - Das ist eine Bedingung an die Speicherschnittstelle, welche von der Komponente importiert wird

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Hier ausreichend: Schnittstelle als der Zugriffspunkt der Komponente
  - Zugriffspunkt = spezieller Dienst samt Kontrakt, der von der Komponente angeboten wird
  - Kontrakt als Interaktions-Basis der sonst unabhängigen Seiten Komponenten-Entwickler und Komponenten-Nutzer
- mehrere Schnittstellen = mehrere Zugriffspunkte = verschiedene Klienten
  - ökonomischer Grund: Skalen-Effekt
- ein Zugriffspunkt oder die Summe der Zugriffspunkte kann über Markterfolg entscheiden
  - wenn keiner der beiden Effekte erzielt werden kann, dann lohnt es nicht, die Software in Komponentenform zu bringen
  - Komponenten ohne Markt können im Rahmen einer bereits eingesetzten Komponenten-Plattform als Lückenschluss sinnvoll sein
    - Skaleneffekt indirekt: Komponente in Kombination mit der Plattform

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Neben den Schnittstellen muss für Komponenten auch klar sein, welche Umgebungsbedingungen für ihre Entfaltung erforderlich sind
  - Spezifikation der Anforderung an die Lokalisierungs-Umgebung
    - auch als Kontext-Abhängigkeit bezeichnet
  - enthält:
    - Komponenten-Modell = Spezifikation der Kompositions-Regeln
    - Komponenten-Plattform = Spezifikation der Regeln für Entpackung, Installation und Aktivierung von Komponenten
- heute existieren mehrere Komponentenwelten nebeneinander
  - sind selbst intern wieder fragmentiert nach Computer- und Netzwerk-Plattformen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Wie „fett“ soll eine Komponente sein?
  - optimal, aber unreal: „richtige Schnittstellenmenge“ und keine Kontext-Abhängigkeit
  - maximal: „fette“ Komponente, die alle benötigten Dienste mitbringt (=Applikation, grobkörnig)
  - minimal: Auslagern aller bis auf die zentrale Funktionalität (=Klasse, feinkörnig)
    - „Maximizing reuse minimizes use.“
    - Grund: Explodierende Kontext-Abhängigkeit
    - würde nur unter statischen Entwicklungsbedingungen funktionieren
    - Beispiel: Linux-Probleme mit Bibliotheksversionen
  - praktisch ist hier ein je ausgewogenes Mittel zu finden
- Je detaillierter Normierung und Standardisierung, desto schlankere Komponenten sind möglich
  - Standardisierung ist in vertikalen Marktsegmenten (funktional) eher möglich als in horizontalen, aber wegen der geringen Marktgröße schwieriger