Vorlesung
Software aus Komponenten

2. Grundlagen

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2004/05

Noch eine Komponentendefinition

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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-Beschreibung

- Muss die Verwendung der Komponente in einem Produktiv-System genau beschreiben
  - Aspekte:
    - Schnittstellen im engeren Sinne
    - Verteilung, 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
  - Schnittstelle als die einzige Einrichtung, über welche Entwickler und Nutzer als unabhängige Parteien verbunden sind.

Schnittstellen und Markt

- 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

Kontext-Abhängigkeit

- 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 Verteilung, Installation und Aktivierung von Komponenten
- heute existieren mehrere Komponentenwelten nebeneinander
  - sind selbst intern wieder fragmentiert nach Computer- und Netzwerk-Plattformen

Komponenten-Gewicht

- 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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Direkte und indirekte Schnittstellen

- Direkte Schnittstelle: Schnittstelle der Komponente selbst
  - meist prozeduraler Natur
- Indirekte Schnittstelle: Schnittstelle von Objekten, die in der Komponente erzeugt werden
  - meist objektorientierter Natur
- Vereinheitlichung durch Einführung eines statischen Objekts möglich
  - typischer Ansatz von OO Sprachkonzepten
- Überladung indirekter Schnittstellen und späte Bindung
  - Provider des Dienstes hängt vom Objekt ab
  - Derselbe Dienst kann über dieselbe Schnittstelle innerhalb desselben Komponenten-Kontexts von unterschiedlichen Anbietern kommen

Schnittstellen und Versionen

- Problem: Schnittstellen können ihr Verhalten zwischen Versionen wechseln
- Management traditionell über Versionsnummern
  - Komponente als unteilbare Einheit => Versionsnummern nur für ganze Komponenten
- Versions-Information in Import- und Exportschnittstellen
  - direkte Schnittstellen: Abfrage zur Bindungszeit, also (nur) vor dem ersten Schnittstellenaufruf
  - indirekte Schnittstellen: Abfrage vor jedem Aufruf erforderlich
    - Alternative: Integration ins Management der Objektidentität
- Problem der Versions-Information, wenn eine Objektreferenz Komponentengrenzen überschreitet
  - Objekt bietet eigene Dienste an
    - etwa Rückgabe in einem bestimmten Format
  - Objekt nutzt Dienste anderer => dynamische Versionskontrolle

Schnittstellen und Versionen

- Schnittstellenversionen müssen klar als kompatibel oder klar als veraltet (deprecated) deklariert werden können
- Ansatz der unveränderlichen Schnittstellen-Spezifikation (immutable interfaces)
  - Neue Versionen nur als neue Schnittstellen
  - Veraltete Schnittstellen werden einfach nicht mehr unterstützt
  - Beispiel: COM = Component Object Model von MicroSoft
- Veränderbare Schnittstellen-Spezifikationen:
  - klares Kompatibilitäts-Konzept erforderlich
  - Installation verschiedener Komponentenversionen in derselben Umgebung kann erforderlich sein.
  - Unterscheidung zwischen Komponenten, die immer in der aktuellsten Version verwendet werden können und Komponenten, die nur in der ursprünglich installierten Version verwendet werden können
    - wird im Rahmen der CLR = Common Language Runtime verfolgt

Schnittstellen-Kontrakt

- Schnittstellen-Spezifikation als Kontrakt zwischen
  - Nutzer der Funktionalität einer Schnittstelle und
  - Anbieter der Implementierung dieser Schnittstelle
- verbreiteter Zugang auf technischer Ebene: durch Vor- und Nachbedingungen (Hoare-Kalkül: {V} P {N})
  - Nutzer sichert die Vorbedingungen V
  - Anbieter sichert dann Nachbedingung N
  - Problem: Sichert funktionale Eigenschaften, aber weder Performanz von P noch Termination überhaupt
- heute üblich: auch nicht-funktionale Aspekte im Kontrakt erfassen
  - Beispiel: Service Level Agreement
    - enthält Qualitätsaussagen für den Betrieb wie Verfügbarkeit, Fehlerrate, Datensicherheit etc.
  - Konsequenzen im Einsatz sind ähnlich gravierend wie funktionale Fehler

- „undokumentierte Features“
  - Auf Komponenten-Verhalten kann jenseits der Spezifikation auch aus Beobachtung des laufenden Betriebs geschlossen werden
  - typisches Ergebnis eines „Debugging“-Prozesses bei Fehlersuche
  - kleine Fehler sind ökonomischer auf der Nutzerseite als auf der Anbieterseite zu beheben
  - Open-Source-Ansatz

Vererbung als Prinzip

- Drei wesentliche Facetten
  - Subklassen: Vererbung von Implementations-Fragmenten
    - Implementations-Vererbung
    - Java: *extends*
  - Subtypen: Vererbung von Kontrakt-Fragmenten
    - Schnittstellen-Vererbung
    - Java: *implements*
  - Substituierbarkeit: Vererbung funktionaler Eigenschaften

- Mehrfach-Vererbung
  - Kein Problem auf Schnittstellenebene
  - Diamanten-Problem auf Implementationsebene
    - auf der Ebene der Zustände (Attribute – Referenz oder Kopie?)
      - Referenz bricht Kapselung
    - auf der Ebene der Methoden

Das Problem der fragilen Basisklasse

- Fragestellung: Was passiert in einer Vererbungsrelation, wenn die Basisklasse durch eine neue Version ersetzt wird?
  - syntaktische Dimension: Wie sieht es mit der Binärkompatibilität von neuer Basisklasse und alten Kindklassen aus?
    - Muss die Kindklasse recompiliert werden?
    - Nein, selbst bei Verschiebungen in der Vererbungshierarchie (Restrukturierung) oder Schnittstellen-Erweiterungen
    - Kann zur Ladezeit über die Dispatch-Tabellen der Klassen behandelt werden
  - semantische Dimension: Die Implementierung der Subklasse nimmt Bezug auf implementatorische (semantische) Details der Basisklasse
    - mit einer neuen Version der Basisklasse kann diese Voraussetzung hinfällig sein.