

- Customer: Could I have a look at the reader-writer package you have in the window?
- Server: Certainly. Would you be interested in this robust version—proof against abort? Or we have this slick version for trusty callers. Just arrived this week.
- Customer: Well—it's for a cooperating system so the new one sounds good. How much is it?
- Server: It's 250 Eurodollars but as it's new there is a special offer with it—a free copy of this random number generator and 10% off your next certification.
- Customer: Creat. Is it validated?
- Server: All our products conform to the highest standards. The parameter mechanism conforms to ES98263 and it has the usual international multitasking certificate.
- Customer: OK, I'll take it.
- Server: Will you take it as it is or shall I instantiate it for you?
- Customer: As it is please. I prefer to do my own instantiation.

– John Barnes, 1998

Komponentenmodelle

- Sicherstellen der Interoperabilität
- Standards für

Schnittstellen:

- Schnittstellenbeschreibung
- spezielle Schnittstellen
- Interaktion zwischen Komponenten

Informationen zur Verwendung:

- Namensregeln
- Individualisieren
- Zugriff auf Metadaten

Einsatz:

- Verpackung
- Dokumentation
- Evolutionsunterstützung

Beispiele für Komponentenmodelle

im weiteren Sinn:

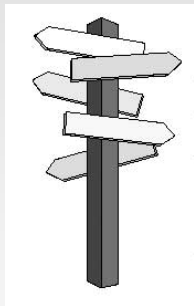
- Anwendungen in einem Betriebssystem
- Plugins
- Verbunddokumente (Office Dokumente mit OLE, HTML)

im engeren Sinn: CORBA, COM, JavaBeans

Eigenschaften erfolgreicher Komponentenmodelle

- Infrastruktur mit guter Basisfunktionalität
- Komponenten werden von unterschiedlichen Herstellern angeboten und von Kunden eingesetzt
- Komponenten von verschiedenen Anbietern arbeiten in einer Installation zusammen
- Komponenten haben eine Bedeutung für den Klienten

- meist durch Komponentenmodelle als Schnittstelle festgelegt
- Anbieter der Komponentenmodelle bietet auch diese Komponenten an
- besonders wichtig für verteilte Systeme
- Beispiele:
 - Verzeichnisdienst
 - Persistenz
 - Nachrichtendienst
 - Transaktionsmanagement
 - Sicherheit



Teil III

Schnittstellen

Schnittstellen

- ermöglichen:
 - Zusammenarbeit zwischen fremden Komponenten
 - Austauschbarkeit (Anbieter und Benutzer)
 - Identifizierung der Abhängigkeiten
- interner Zustand nichtöffentlich (alle Zugriffe über Schnittstelle)
- Qualität von höchster Bedeutung

Schnittstellen

- ermöglichen:
 - Zusammenarbeit zwischen fremden Komponenten
 - Austauschbarkeit (Anbieter und Benutzer)
 - Identifizierung der Abhängigkeiten
- interner Zustand nichtöffentlich (alle Zugriffe über Schnittstelle)
- Qualität von höchster Bedeutung
- eine Komponente kann Serviceprovider für mehr als eine Schnittstelle sein (provides)
- eine Komponente kann den Service anderer Schnittstellen benötigen (requires)

Schnittstellen

- ermöglichen:
 - Zusammenarbeit zwischen fremden Komponenten
 - Austauschbarkeit (Anbieter und Benutzer)
 - Identifizierung der Abhängigkeiten
- interner Zustand nichtöffentlich (alle Zugriffe über Schnittstelle)
- Qualität von höchster Bedeutung
- eine Komponente kann Serviceprovider für mehr als eine Schnittstelle sein (provides)
- eine Komponente kann den Service anderer Schnittstellen benötigen (requires)
- späte Integration → späte Bindung → Indirektion
- direkte (prozedural) und indirekte (Objekt-) Schnittstellen
- beschrieben durch Meta-Information zur Laufzeit, Interface Description Language (IDL) oder „direkt“ (Java)

Beispiel (CORBA-IDL)

```
module Bank {  
    typedef long pin_t;  
    enum KontoFehlerTyp {  
        UngenuegendeKontodeckung  
    };  
};
```

Beispiel (CORBA-IDL)

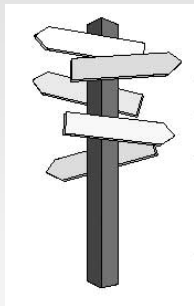
```
module Bank {  
    typedef long pin_t;  
    enum KontoFehlerTyp {  
        UngenuegendeKontodeckung  
    };  
  
    exception KontoException {  
        KontoFehlerTyp typ;  
        string beschreibung;  
    };  
};
```

Beispiel (CORBA-IDL)

```
module Bank {  
    typedef long pin_t;  
    enum KontoFehlerTyp {  
        UngenuegendeKontodeckung  
    };  
  
    exception KontoException {  
        KontoFehlerTyp typ;  
        string beschreibung;  
    };  
  
    interface Konto {  
        readonly attribute string name;  
        readonly attribute long kontoStand;  
  
        boolean isValidPin(in pin_t pin);  
        void abheben(in long betrag) raises(KontoException);  
        void einzahlen(in long betrag);  
    };  
};
```

- Schnittstelle als Vertrag zwischen Anbieter und Benutzer des Services
- Anbieter:
 - über Funktionalität: z.B. als Vor- und Nachbedingungen
 - über nicht-funktionale Anforderungen (Service-Level, Ressourcen); z.B. Standard Template Library für C++
 - Darstellung:
 - informal als Text
 - formaler z.B. durch temporale Logik (um Terminierung zuzusichern) oder mit OCL
- Kunde:
 - Vermeidung von speziellen Eigenschaften einer bestimmten Implementierung (d.h. nur Vertrag benutzen)

- Problem: sowohl Schnittstelle als auch Implementierung ändern sich
→ unterschiedliche Versionen nicht vermeidbar
- Ziel: Entscheidung, ob kompatibel oder nicht
 - zusätzlich noch Unterstützung für einen Bereich von Versionen (sliding window)
- Lösungen (Lösungen?):
 - unveränderliche Schnittstellen
 - Schnittstellen dürfen sich ändern, aber nur nach Regeln (z.B. Parametertyp darf verallgemeinert werden)
 - Ignorieren des Problems:
 - abwälzen auf tiefere Schicht
 - immer neu kompilieren



Teil IV

Beschaffung und Entstehung

- Annahme: großer Markt von Komponenten
- Suche: Komponenten und funktionale Anforderungen werden klassifiziert
- Qualitätskriterien für Auswahl:

- Annahme: großer Markt von Komponenten
- Suche: Komponenten und funktionale Anforderungen werden klassifiziert
- Qualitätskriterien für Auswahl:
 - funktionale und nicht-funktionale Anforderungen
 - Schnittstellenbeschreibung
 - Abhängigkeiten und Kompatibilität
 - Vertrauenswürdigkeit, Überlebenschance des Anbieters, Garantien und Wartungszusicherung
 - Preis und Zahlungsart
 - ...

Entstehung von Komponenten

- meist aus bestehender Software
- Anpassung notwendig, um Wiederverwendbarkeit zu erreichen
- ist Investition ökonomisch sinnvoll?
 - Größe des Einsatzgebiets
 - bislang angebotene Funktionalität
 - potentieller Grad der Wiederverwendung

Entstehung von Komponenten

- meist aus bestehender Software
- Anpassung notwendig, um Wiederverwendbarkeit zu erreichen
- ist Investition ökonomisch sinnvoll?
 - Größe des Einsatzgebiets
 - bislang angebotene Funktionalität
 - potentieller Grad der Wiederverwendung

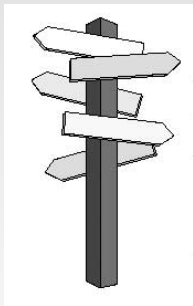
Balance zwischen

großen Komponenten

- bieten mehr Service an
- weniger Abhängigkeiten
- schneller, da keine cross-context Aufrufe

kleinen Komponenten

- verständlicher
- billiger für den Benutzer
- mehr Freiheit für den Benutzer
- mehr Benutzer



Teil V

Implementierungsaspekte

- defensives Programmieren, da keine Integrationstests
- wiederverwendbar machen:
 - entferne anwendungsspezifische Methoden
 - generalisiere Namen
 - füge Methoden hinzu, um Funktionalität zu vervollständigen
 - führe konsistente Ausnahmebehandlung ein
 - füge Möglichkeiten hinzu, die Komponenten an verschiedene Benutzer anzupassen
 - binde benötigte Komponenten ein, um die Unabhängigkeit zu erhöhen