Vorlesung
Software aus Komponenten

2. Grundlagen

apl. Prof. Dr. Hans-Gert Gräbe
Wintersemester 2007/08

(Vertretung Herr Berger)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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?
  - Wenn nur Methoden vererbt werden: im Prinzip nein
  - Selbst bei Verschiebungen in der Vererbungshierarchie (Restrukturierung) oder Schnittstellen-Erweiterungen nicht
  - Grund: Methodenbindung erfolgt zur Laufzeit über die Dispatch-Tabellen der Klassen

  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.
  - auch durch Recompilieren nicht aus der Welt zu schaffen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Basisklasse implementiert read/write auf abstrakter Ebene

```
abstract class Text {
    private char[] text = new char[1000]; // Textbuffer
    private int used = 0; // Position des letzten Textzeichens
    private int caret = 0; // Position des Kursors

    void setCaret(int pos) { caret=pos; }
    int caretPos() { return caret; }
    ...
    void write(int pos, char ch) {
        for (int i=used; i>pos; i--) { text[i]=text[i-1]; }
        used++; text[pos]=ch;
        if (caretPos()>=pos) setCaret(caret+1);
    }
    abstract int posToX(int pos);
    abstract int posToY(int pos);
    abstract int posFromCoords(int x, int y);
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Subklasse implementiert View mit Kursor

```
class SimpleText extends Text {
    private int cacheX = 0; private int cacheY = 0;

    void setCaret(int pos) { // überschriebene Methode
            int old = caretPos();
            if (old != pos) { hideCaret(); super.setCaret(pos); showCaret(); }
    }
    int posToX(int pos) {...}
    int posToY(int pos) {...}
    int posFromCoords(int x, int y) {...}
    void hideCaret() { ... }
    void showCaret() { ... }
}
```

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Neue Version der Klasse Text:

```
abstract class Text {
    ...
    void write(int pos, char ch) {
        for (int i=used; i>pos; i--) { text[i]=text[i-1]; }
        used++; text[pos]=ch;
        if (caret >= pos) caret++;
    }
}
```

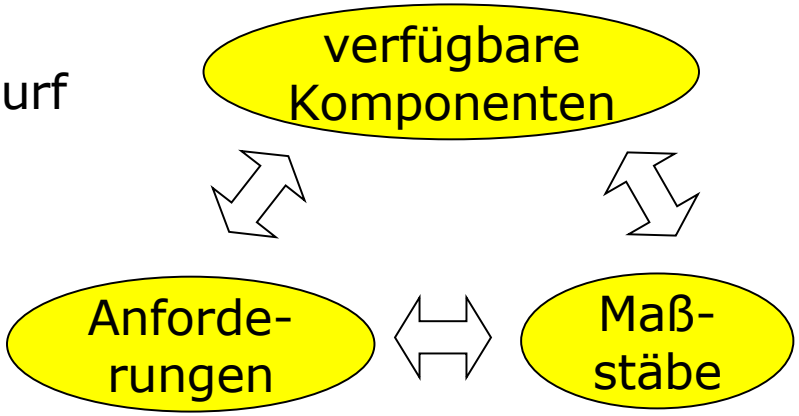
Effekt: Nach *write* steht der grafische Kursor an der falschen Position, weil die Implementierung von *SimpleText* sich darauf verlassen hat, dass *Text* die Variable *caret* stets nur über *setCaret* manipuliert.

Wechsel der inneren Implementierung des Anbieters kann den Nutzer korrumpieren, ohne den Kontrakt zu verletzen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten abgrenzen

Problem: Wie ist ein kompletter Entwurf in Komponenten zu partitionieren?



Prinzipien:

- Modularität und Kapselung
- Abhängigkeiten zwischen K. sind expliziert
- Mehrere Abstraktionsebenen, hierarchische Strukturierung
- natürliche Zuordnung von Verantwortlichkeiten
- Migrations-Erfordernisse vorab berücksichtigen

- 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

Typische Betrachtungsansätze

(führen zu unterschiedlicher Granularität)

- **K. als Einheit der Abstraktion**

  Ansatz: white box black box

  Entwurfsexpertise kapseln (design expertise ready for use)

  Theoretische Einschränkung der Vielfalt in der aktuellen Ebene ist Basis für größere Vielfalt in der nächsthöheren Ebene

- **K. als Einheit der Kostenrechnung**

  wichtig in größeren industriellen Kontexten, um Projektentwicklungskosten verfolgen zu können

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- **K. als Einheit des Managements**

  oft zu klein, Management auf der Ebene von Subsystemen, die mehrere Komponenten zusammenfassen

  - etwa auf der Ebene der Server

- **K. als Einheit der Analyse**

  Analyse an vielen Stellen im Komponentenlebenszyklus erforderlich (Spezifikationsprüfung, Tests, Re-Engineering)

  Kopplung zwischen Einheiten bestimmt, wie weit eine individuelle Analyse zweckmäßig bzw. überhaupt möglich ist

  Regel: Einheiten für Analyse so klein wie möglich

  Regel: streng statische hierarchische Grenzen (Moduln, Subsysteme) erleichtern die Analyse

- **K. als unabhängig compilierbare Einheit**

  Compilierbarkeit und Analyse sind eng verbunden
  - white box: Analyse
  - black box: Compilierbarkeit

  sinnvolle Einheiten: Moduln oder Klassen

  Globale Optimierung?
  - Antwort 1: Einheiten möglichst groß wählen
  - Antwort 2: Optimierung zwischen Komponenten, vielleicht sogar erst nach der Lokalisierung
    - muss im Komponentenkonzept und der Komponentenbeschreibung verankert sein

- **K. als Auslieferungseinheit**

  Bündel der technischen und wirtschaftlichen Aspekte

  Management (Service, Wartung, Schulung, Updates, ...) treibt den Preis in die Höhe
  - betriebswirtschaftliche Bedeutung jenseits der (geringen) Replikationskosten

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- **K. als Packungs-Einheit**

  Entpackung (deployment) = Prozess der Vorbereitung der Komponente auf den Einsatz in einer speziellen Umgebung (Lokalisierung)

  - wurde lange nicht als separater Schritt betrachtet
  - Prozess der Anbindung an eine spezielle Komponentenplattform

  Konfiguration = Einstellung spezieller Eigenschaften der Komponente für den konkreten Einsatz

  Installation = plattformspezifische Aktivität, mit der eine entpackte Komponente für die Nutzung in einer speziellen Hardware-Konfiguration verfügbar gemacht wird, die von der Plattform unterstützt wird.

  - Zeit, in der auch kritische Tests ausgeführt werden, die vor dem eigentlichen Betrieb erfolgen müssen (etwa Integritätstests)

  für alle drei Aktivitäten müssen entsprechende Beschreibungen erstellt werden

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- **K. als Einheit der (Auseinandersetzung um) Fehlersuche**

  Problem: Was ist (u.a. wer haftet?), wenn ein aus Komponenten zusammengebautes Produktivsystem fehlerhaft arbeitet?

  Problem der Lokalisierung von Fehlern (und damit Verantwortlichkeiten)

  - besonders schwierig wenn Objektreferenzen die Komponentengrenzen verlassen

  vitale Regel: Fehler müssen in den verursachenden Komponenten bleiben (bug containment)

  - typische nicht-lokalisierbare Fehler: Speicherzugriffsfehler

  Folge: Ausnahmebehandlungen müssen in der Regel innerhalb einer Komponente bleiben

  - Ausnahmen davon sind im Komponentenkontrakt zu fixieren

    Komponente als Einheit der Fehlerbehandlung

- **K. als Einheit der Erweiterung**

  Beispiel: K. implementiert eine konkrete (standardisierte) Schnittstelle

  Objekte in einer Erweiterungshierarchie sind gewöhnlich enger gekoppelt als andere
  - „friends"-Mechanismus, „protected"-Schnittstelle
  - sind Ersatz für Mechanismen der Kapselung mehrerer Objekte

  unabhängige Erweiterungen und deren Koexistenz

  eine Einheit der Analyse darf nicht in mehrere Erweiterungseinheiten zerlegt werden
  - gemeinsame Analyse hat oft kontextuelle Abhängigkeiten zwischen den Teilen zur Folge

- **K. als Einheiten der Instanziierung**

  Einheit der Instanziierung sind Objekte

  also: kein sinnvolles Kriterium für Komponentenabgrenzung

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- **K. als Einheit des Ladens**

  oft wird beim ersten Einsatz einer Funktionen erst entsprechender Code geladen.

  benötigt Mechanismen des dynamischen Ladens (etwa DLL)

  typisch werden dabei gleich mehrere zusammenhängende Klassen geladen

  Komponente als Einheit des Ladens sinnvoll

  Kontrolle der Version
  - Problem der fragilen Basisklasse
  - Maximale Flexibilität, wenn Versionskontrolle auf der Ebene von Schnittstellen oder sogar Methoden
  - Java: Versions-Check erst zur Laufzeit
    - ist problematisch, da Inkompatibilitäten erst mitten in der Programmausführung entdeckt werden

  Kontrolle der Erfüllbarkeit der Importrelationen

  Kontrolle von Namenskollisionen
  - Lösung: Hierarchisches Namensraum-Konzept
  - Problem der Namenskollision: Versionen derselben Komponente

- K. als Einheit des Ladens (Fortsetzung)

  Konsistenz bereits geladener Komponenten muss erhalten bleiben
  - Konsistenz muss dazu lokale Eigenschaft sein
  - schließt z.B. globale Typprüfungsansätze aus

- **K. als Einheit der Lokalität**

  Problem im Kontext verteilter Systeme: was befindet sich (lokal) auf welchem Rechner
  - typisch sind hierarchische Konzepte des Clusterns von Rechnern
    - SAN, LAN, WAN, Internet (und weitere Zwischenstufen)
    - unterschiedliche Kommunikationszeiten und -kosten

  Tradeoff: minimale Kommunikationskosten vs. maximale Ressourcennutzung

  hohe Kopplung zwischen Objekten aus derselben Komponente

  Komponente sollte nicht auf verschiedene Prozesse oder Maschinen verteilt sein
  - Bündelung von Zugriffen auf Objekte über Prozessgrenzen hinweg
    - ein komplexer statt mehrerer einfacher Zugriffe

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten-Modelle

3. Grundlagen: Kommunikationskonzepte
4. Erste Komponentenansätze
5. OMG und CORBA – der geschäftsprozesszentrierte Ansatz
6. Sun und Java – der webzentrierte Ansatz
7. Microsofts und .NET – der dokumentenzentrierte Ansatz

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Interprozess-Kommunikation (IPC) auf OS-Ebene

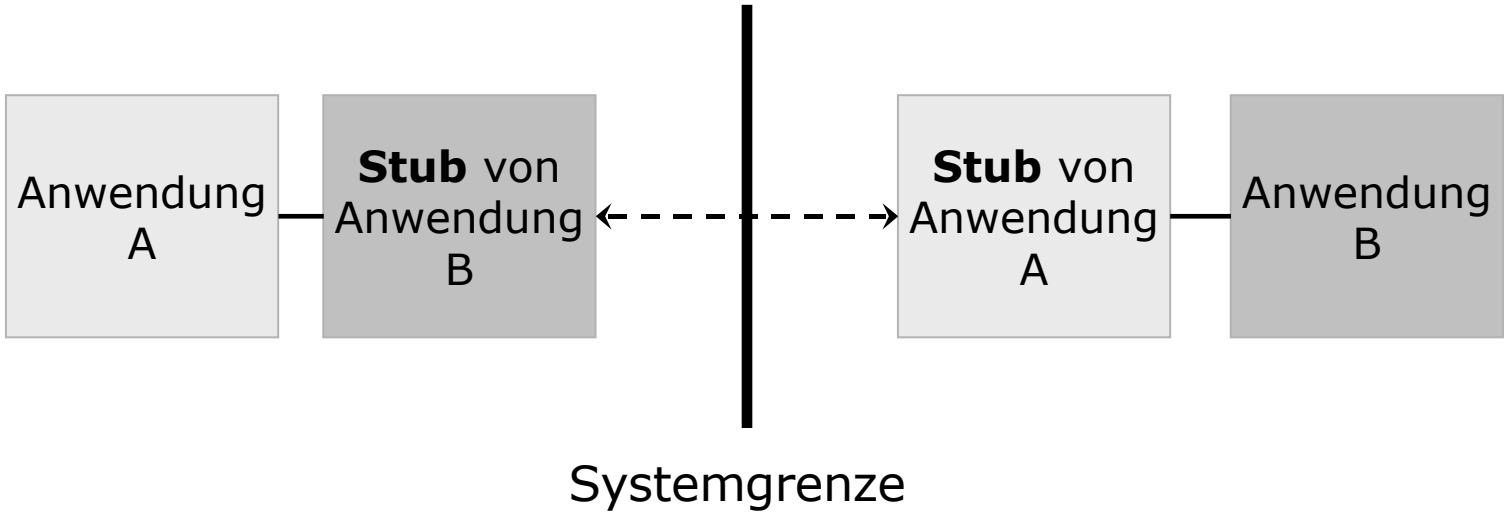
- Charakteristika
  - kaum plattformübergreifend standardisiert
  - nicht Teil des von-Neumann-Modells
  - Prozess = virtueller Rechner auf physischem Host
- IPC-Modelle: Dateien, Pipes, Sockets, Semaphore, shared memory
  - außer Sockets keins so weit standardisiert, dass es plattformübergreifend eingesetzt werden könnte.
  - außer shared memory skalierbar und internetfähig
  - operiert auf Bitebene -> zu kompliziert für komplexe Anwendungen

IPC operiert auf Bitebene und ist deutlich zu kompliziert für komplexe Anwendungen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Remote Procedure Calls (RPC, 1984)

- Ansatz: Stubs, die auch entfernte Prozeduraufrufe lokal aussehen lassen
  - Aufgabe des Stub: Serialisierung bzw. Deserialisierung des Prozeduraufrufs und der Aufrufparameter unter Beachtung von plattformabhängiger Byte-Kodierung, Zahldarstellung, ...



- Nutzung leichtgewichtiger RPC zur IPC auf derselben Maschine (Windows NT)
- Vorteil: einheitliches Abstraktionsniveau für alle Kommunikationserfordernisse (innerhalb eines Prozesses, zwischen Prozessen, zwischen Computern)
- Nachteile:
  - Versteckte Kommunikationskosten (Unterschied um Faktor $10...10^4$), Client kann nicht unterscheiden, ob lokaler oder entfernter Aufruf
  - blockierendes Konzept
  - Umgang mit Versionierung und Evolution von Komponenten vollkommen unklar

Das RPC-Konzept bildet zusammen mit dynamisch linkbaren Bibliotheken (DLL) die Basis für das einfachste Komponenten-Framework (und ist das heute am weitesten verbreitete).

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Distributed Computing Environment (DCE)

- Standard der Open Software Foundation für RPC auf heterogenen Plattformen (http://www.opengroup.org/dce, aktuelle Version 1.2.2)
- **Interface Definition Language** (IDL)
  - zur Standardisierung des Absetzens von RPC
  - genaue Angabe von Architekturspezifika erforderlich
    - etwa: int = 32-bit low-endian Zweierkomplementdarstellung
- **Universally Unique IDentifiers** (UUID)
  - Standard zur global eindeutigen Bezeichnung und Identifikation von Computern, Prozessen, Prozeduren und Daten
  - maschinenorientierte Bitdarstellung, deshalb wird zur menschenlesbaren Beschreibung mit Aliasen gearbeitet

DCE ist ein Standard, um (u.a.) RPC zwischen heterogenen Plattformen konsistent einzusetzen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Objekte und Methoden

- RPC ist ein statisches Aufrufkonzept
- Besonderheit von Methoden gegenüber Prozeduren:
  - werden dynamisch an Hand der Charakteristika des Objekts (=Instanz seiner Klasse) ausgewählt
    - erst nach dieser Auswahl kann der RPC-Mechanismus greifen
    - Klassen müssen dazu genügend (binär kodierte) Information bieten, die durch Introspektion zur Laufzeit abgefragt werden kann
  - Objektreferenzen als Aufrufparameter
    - Keine automatischen Objektkopien

Der RPC-Ansatz ist deutlich aufzubohren, wenn mit Objekten und Methoden mit laufzeitabhängigem Verhalten umgegangen werden soll.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Erweiterung des RPC-Konzepts

- Verwendung von Funktionsvariablen, die zur Laufzeit mit einem Funktionszeiger als Wert belegt werden
- Virtuelle Methodentabellen (VMT)
  - Beispiel: COM‘s dispatch tables (Microsoft)

oder Spezielle Laufzeitumgebung (virtual machine, VM)

- übernimmt Management von *Methoden*-Aufrufen
  - SOM (IBMs System Object Model)
  - Java (Java virtual machine)
  - .NET common language runtime (CLR)
- braucht spezielle Mechanismen, um **über die Grenzen der VM hinaus** mit Komponenten zu kommunizieren

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Über RPC-Mechanismus hinaus gehende Fragen, die ein objektorientiertes Kommunikationskonzept beantworten muss

1. Wie werden Schnittstellen spezifiziert?
2. Wie werden Objektreferenzen behandelt, wenn der lokale Bereich verlassen wird?
3. Wie werden Dienste aufgefunden und bereitgestellt?
4. Wie wird die Evolution von Komponenten gehandhabt? (Versionsmanagement)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Schnittstellenspezifikation

- **Definition:** Interface ist ein abstrakter Datentyp
  - Sammlung von Operationsbezeichnern mit ihren Signaturen
  - Signatur = Typ und Aufrufmodi der Parameter

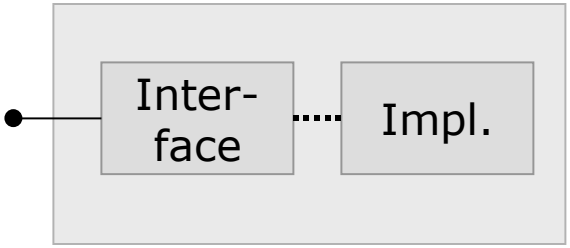
- **Schnittstellen-Beschreibung** durch IDL
  - mehrere Standards koexistieren (insb. OMG IDL und COM IDL)
  - Java und CLR: Keine IDL, sondern sprachspezifisches Meta-Datenformat, das auf jede der IDL abgebildet werden kann
    - dazu sind entsprechende Abbildungen zu spezifizieren
      - *Java to IDL language mapping specification* (Version 1.3 vom Sept. 2003, siehe http://www.omg.org)
    - **Umgekehrt:** Java-Werkzeug **idlj** erzeugt aus einer (OMG) IDL-Beschreibung (u.a.) ein Java *interface*.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Schnittstellen und Implementierungen

Drei wesentlich verschiedene Ansätze:

- 1 Impl. 1 S (CORBA 2, SOM)
  - Objekt = Programmzustand (Kontrollfluss **und** Daten) und Implementierung **seiner** Schnittstelle
  - CORBA-Objektbegriff damit **zwischen** Komponente und Objekt im Sinne der Vorlesung
  - Schnittstelle kann durch Mehrfachvererbung entstanden sein

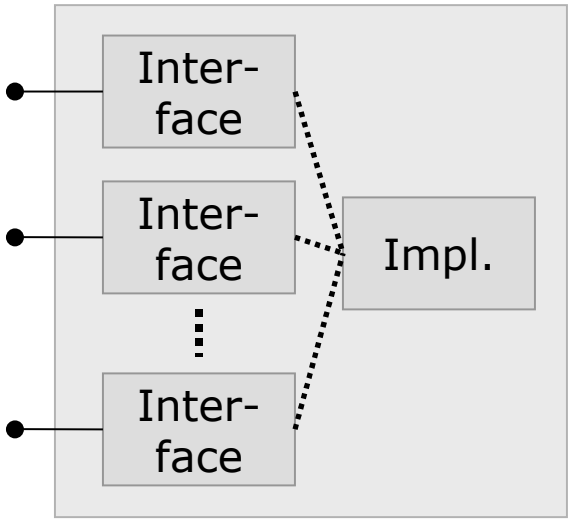


CORBA

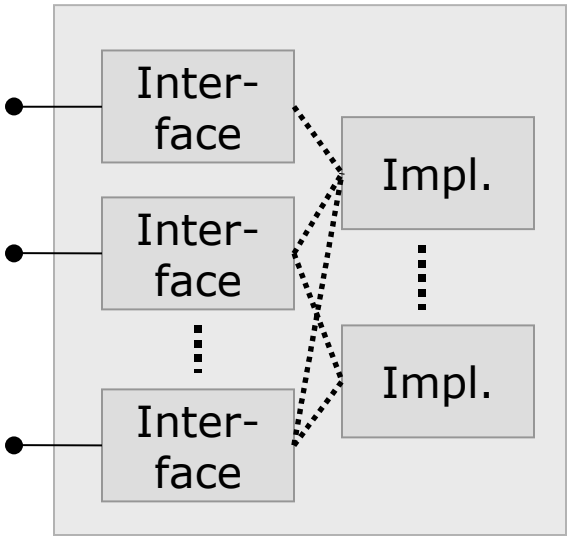
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- 1 Impl. * S (Java, CLR)
- * Impl. * S (COM)

Java



COM



- mehrere Implementierungen derselben Schnittstelle möglich
- Implementierung kann mehr Funktionalität bereitstellen als durch die Schnittstelle definiert

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- CORBA: Traditioneller Objektansatz
  - Jedes Komponente (=Objekt) hat nur ein Interface
  - Mehrfachvererbung möglich
  - Erwartete Schnittstelle darf Subtyp der bereitgestellten sein
    - Zusatzeigenschaften können dynamisch herausgefunden werden

- COM: Komponente hat mehrere Schnittstellen in einer Liste
  - Schnittstellen bedienen mehrere Objekte
  - Interface unveränderbar
    - einmal veröffentlicht -- weder erweiter- noch änderbar
    - aber Schnittstellenliste kann dynamisch erweitert werden
  - einfache Schnittstellenvererbung möglich

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- Java: Klassen können mehrere Schnittstellen implementieren, aber stärker am Vererbungskonzept orientiert
  - Default-Implementierungen von Interfaces durch abstrakte Klassen möglich
  - Klasse kann mehrere Interfaces implementieren, aber nur von einer abstrakten Klasse erben

- Problem der Namenskollision, wenn Methoden aus unterschiedlichen Schnittstellen denselben Namen haben
  - Java: Überladen und Überschreiben
    - qualifizierte Namensgebung ist möglich
  - COM und CLR: unterschiedliche Schnittstellen haben unterschiedliche Namensräume

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Namensgebung und Auffinden von Diensten

- Dienste werden über ihren Namen identifiziert
  - OMG: UUID als Standard der Open Software Foundation (DCE)
    - genügend lange Zeichenkombinationen
  - COM (Microsoft) verwendet modifizierte Version: Global Unique IDentifier (GUID)
    - Namensgebung für Interfaces (IID), Gruppen von Interfaces (categories = CATID) und Klassen (CLSID)
    - CLR: Identität durch private / public-key auf Komponentenebene
  - Java: Eindeutigkeit über zusammengesetzte Pfadnamen (Anlehnung an URL)

- Über den Namen muss wenigstens folgende Funktionalität zur Laufzeit abrufbar sein:
  - Typtest der Schnittstellen
  - Introspektion der Schnittstellen
  - dynamisches Erzeugen neuer Objekte

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Der dokumentenzentrierte Ansatz

- Idee: Nutzer wird nicht mit vielen verschiedenen Applikationen konfrontiert, sondern mit Dokumenten, die aus mehreren Teilen bestehen können. Diese Teile können unterschiedliche Applikationen zur Darstellung benötigen, kennen diese aber selbst.

- Erste Realisierung unmittelbar auf der Ebene von integrierten Textdokumenten
  - Hypercard (Apple)
  - Word mit Visual Basic und VBX (Microsoft)

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Visual Basic

- Dokument besteht aus (mehreren) Formularen
- Formular kann mit Kontrolleinheit ausgestattet sein
- Kontrolleinheiten interagieren über Basic-Skripte

Flexibilität und Produktivität dieses Konzepts führten zur Herausbildung des ersten Komponentenmarkts mit Komponenten etwa zur Tabellenkalkulation oder zur Prozessautomatisierung.

OLE als Weiterentwicklung dieses Ansatzes

- Formulare -> Container für beliebige Anwendungen
- Kontrolleinheit -> Dokumentenserver
- Container können hierarchisch ineinander geschachtelt werden

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Der webzentrierte Ansatz

- Idee: Einbettung von beliebigen Objekten in HTML-Seiten
  - z.B. Java Applets, Form-Bestandteile
- Einheitliche und erweiterbare Darstellung im Browser durch Plugin-Technologie
- Schritt weg vom OLE-Containerkonzept und zurück zum (nicht hierarchischen) Formularansatz von Visual Basic

Aktuelle Entwicklungsrichtungen

- COM (Microsoft)
- CORBA (Object Management Group)
- Java (Sun und inzwischen auch IBM)
- Webservices als lose gekoppeltes Konzept

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Die OMG und CORBA

- Geschichte, Zielstellungen, Entwicklungsetappen
- Architektur
  - Objekte, Servanten, Anwendungen
  - Schnittstellensprache OMG IDL
  - Dynamische Methodenaufrufe (DII)
  - Symmetrie des CORBA-Modells
- Der Object Request Broker (ORB)
- CORBA-Objekte und Objektreferenzen
- CORBA IDL und Datentypen

- Literatur: CORBA Spezifikation 3.0 (Juli 2002)
  - 1154 Seiten pdf-Dokument, siehe http://www.omg.org

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zur Geschichte der OMG (Object Management Group)

- **Ausgangspunkt 1989:** Wie kommuniziert man in einem verteilten OO-System über Sprach- und Plattformgrenzen hinweg?
  - selbst auf derselben Plattform lieferten C++-Compiler inkompatiblen Bytecode, verschiedene Objektmodelle in verschiedenen Programmiersprachen, Plattformunterschiede bei Socket-Kopplung
  - „Deep gaps everywhere"
- im April 1989 von 11 Firmen gegründet
- heute mit ca. 800 Mitgliedern eines der größten Konsortien der Computer-Industrie
  - vor allem Systemanbieter und Anwender objektorientierter Techniken

Zielstellung: „Standardisierung, koste es, was es wolle", um Interoperabilität auf allen Ebenen in einem offenen Markt für „Objekte" zu erreichen.

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Zielstellungen der OMG

- Offene Interoperabilität zwischen einer Vielzahl von Sprachen, Implementierungen und Plattformen
- mehr standardisieren als „binäre" Standards
- Flexibilität statt Binärkompatibilität
  - „teure" Hochsprachenprotokolle
- **Nichtkommerzielle Vereinigung** zur Entwicklung von technisch exakten und in der Praxis realisierbaren Spezifikationen
- **Vereinbarung von Standards und Spezifikationen** der Infrastruktur für verteilte, objektorientierte Anwendungen
- **Aufstellung von Richtlinien** (guidelines) zur Entwicklung von Umgebungen, in denen heterogene Systemen (verschiedene Plattformen, Betriebssysteme u.ä.) zusammenarbeiten können
- Durch standardisierte, objektorientierte Softwarekonzepte die **Entstehung eines Marktes für Komponentensoftware forcieren**

Etappen der Entwicklung von CORBA

CORBA 1 (seit 1991) : **Standardisierung des ORB**

- erste Lösungen, um das Wirrwar zu entflechten
- Ansatz: Vermittlung zwischen Anfragen und Diensten durch einen Object Request Broker (ORB)
- CORBA = Common Object Request Broker Architecture
- **Meilenstein**: Schnittstellen-Definitionssprache (OMG IDL)

CORBA 2 (seit 1995 – 96) : **Interoperationsstandards zwischen ORBs**

- **Meilenstein**: Internet Inter-ORB Protokoll (IIOP)
- muss von jeder ORB-Implementierung unterstützt werden
- Für CORBA 2 existiert Vielzahl von Realisierungen verschiedener Anbieter und für verschiedene Plattformen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Etappen der Entwicklung von CORBA (Fortsetzung)

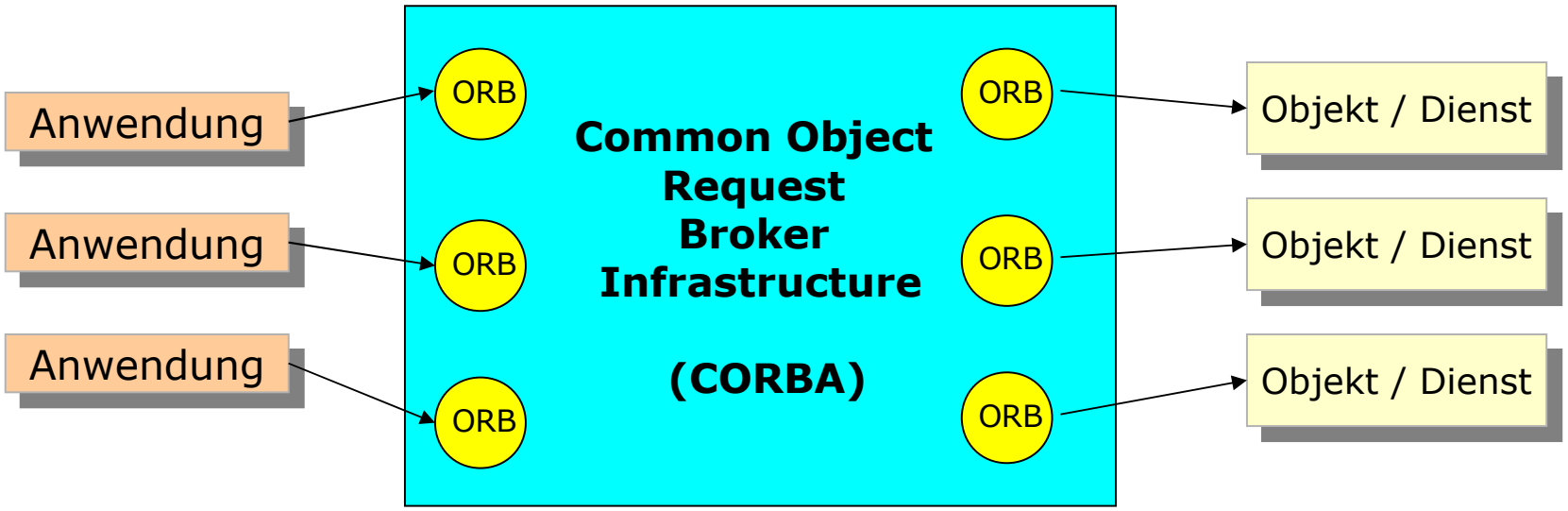
CORBA 3 (12/2002) : **Komponenten- und Systemintegration**

- Höhere Abstraktionsebene
- neue Sprachebenen zur Beschreibung von Komponenten-Eigenschaften
- seit 1998 in der Entwicklung, aber als Ganzes erst Ende 2002 freigegeben
  - CORBA 2.3 ... 2.6 (2001) : Freigabe verschiedener Standards, auf die man sich auf dem Weg zu CORBA 3 zwischenzeitlich geeinigt hatte
- **Meilenstein**: CORBA Komponentenmodel (CCM)
  - Version 4.0, April 2006
- aktuelle Version CORBA/IIOP 3.0.3, März 2004 (http://www.omg.org)
- bisher kaum Implementierungen, die CORBA 3 voll unterstützen

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

CORBA besteht im Grunde aus drei wichtigen Teilen:

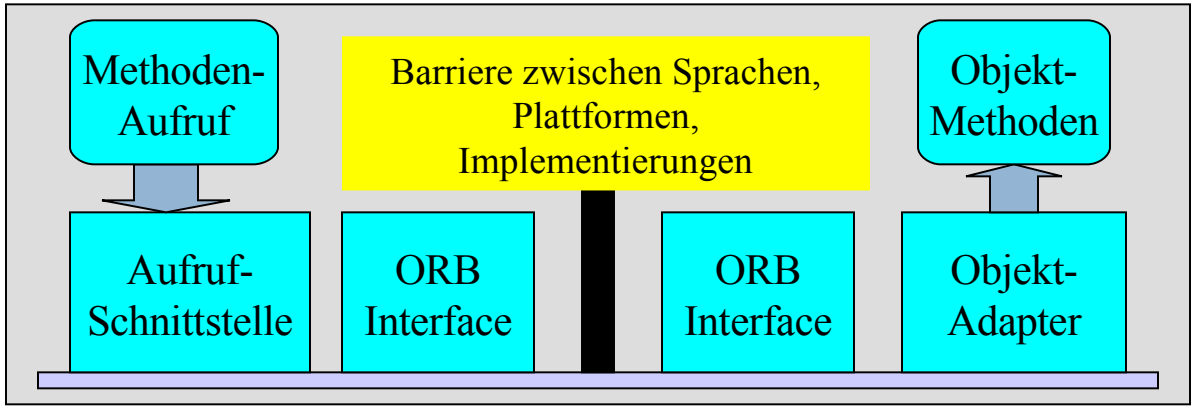
- einer Menge von Aufrufschnittstellen (Invocation Interfaces)
- den Vermittlern (Object Request Brokers – ORBs) als den Schaltstellen der Kommunikation
  - mit einem spezifizierten Protokoll, dem internet inter-ORB protocol IIOP
- einer Menge von Objekt-Adaptern



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Laufzeitbindung von Methodenaufrufen

- Aufrufschnittstelle serialisiert Aufrufargumente
- ORBs suchen Zielobjekt, -methode, organisieren Transport der Argumente
- Objektadapter: dient der Aktivierung des Diensts im Objekt. Deserialisiert Argumente und ruft entsprechende Methode des Zielobjekts auf.



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Wichtige Voraussetzungen

- Schnittstellen müssen in einer einheitlichen Sprache **definiert** werden (**Interface Definition Language - OMG IDL**)
  - wesentlicher Bestandteil des CORBA-Standards
  - ermöglicht generisches Serialisieren / Deserialisieren

```
module Example {
  struct Date {
            unsigned short Day;
            unsigned short Month;
            unsigned short Year; }

interface Ufo {
            readonly attribute unsigned long ID;
            readonly attribute string Name;
            readonly attribute Date FirstContact;
            unsigned long Contacts ();
            void RegisterContact (Date dateOfContact); }
}
```

Datentyp-Definition

Attribute der Schnittstelle

Methoden der Schnittstelle

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Wichtige Voraussetzungen (Fortsetzung)

- alle Programmiersprachen, die den CORBA-Standard unterstützen, müssen **an OMG IDL gebunden** werden.
  - Mapping von Datentypen,
  - Übersetzung des OMG IDL Operationsformats in das sprachspezifische Aufruf-Format
  - Fehlerbehandlung
  - existieren Anbindungen für C, C++, SmallTalk, Cobol, Java, ...

In OMG IDL beschriebene Schnittstellen werden dann

- mit einem **OMG IDL Compiler** übersetzt
- im **Schnittstellen-Repository** abgelegt
- durch **Methoden der ORB-Schnittstelle** angesprochen

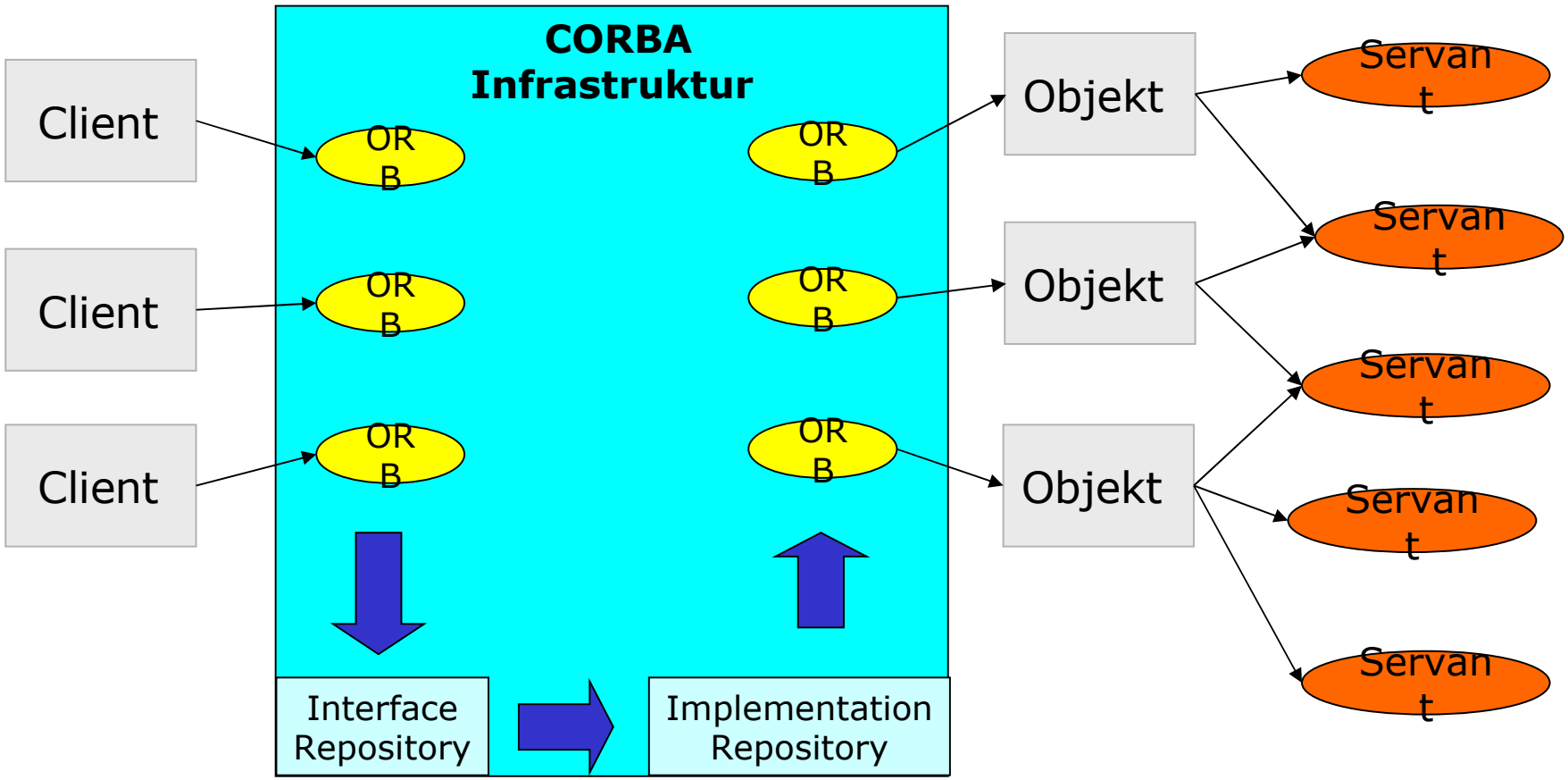
UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Wichtige Voraussetzungen (Fortsetzung)

- Programmfragmente stellen **Implementierungen** für solche Schnittstellen (oder Teile davon) bereit
  - heißen **Objekt-Servanten** (object servant)
  - werden im **Implementations Repository** registriert
  - Servanten werden bei Bedarf geladen und/oder gestartet
  - Objektadapter teilen dem ORB mit, welche Objekte von welchen Servanten bedient werden.
  - Eine Serverumgebung (typ. Prozess) kann mehrere Servanten bedienen.
  - *:* - Beziehung zwischen Objekten und Servanten
  - Objektbegriff hat damit leicht anderen Fokus als in der Vorlesung

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Architektur im Überblick



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Stummel (stubs) und Skelette (skeletons)

- Methodenaufrufe erfolgen über Stummel-Skelett-Prinzip des RPC
  - mit OMG IDL Compiler aus Schnittstellenbeschreibung generierbar
- direkt nur für statische Methodenaufrufe einsetzbar (static invocation interface SII / static skeleton interface SSI)