Vorlesung
Software aus Komponenten

3. Komponentenmodelle

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten-Modelle

1. Grundlagen: Kommunikationskonzepte
2. OMG und CORBA – der geschäftsprozesszentrierte Ansatz
3. Sun und Java – der webzentrierte Ansatz
4. Microsofts und .NET – der dokumentenzentrierte Ansatz

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Interprozess-Kommunikation (IPC) auf OS-Ebene

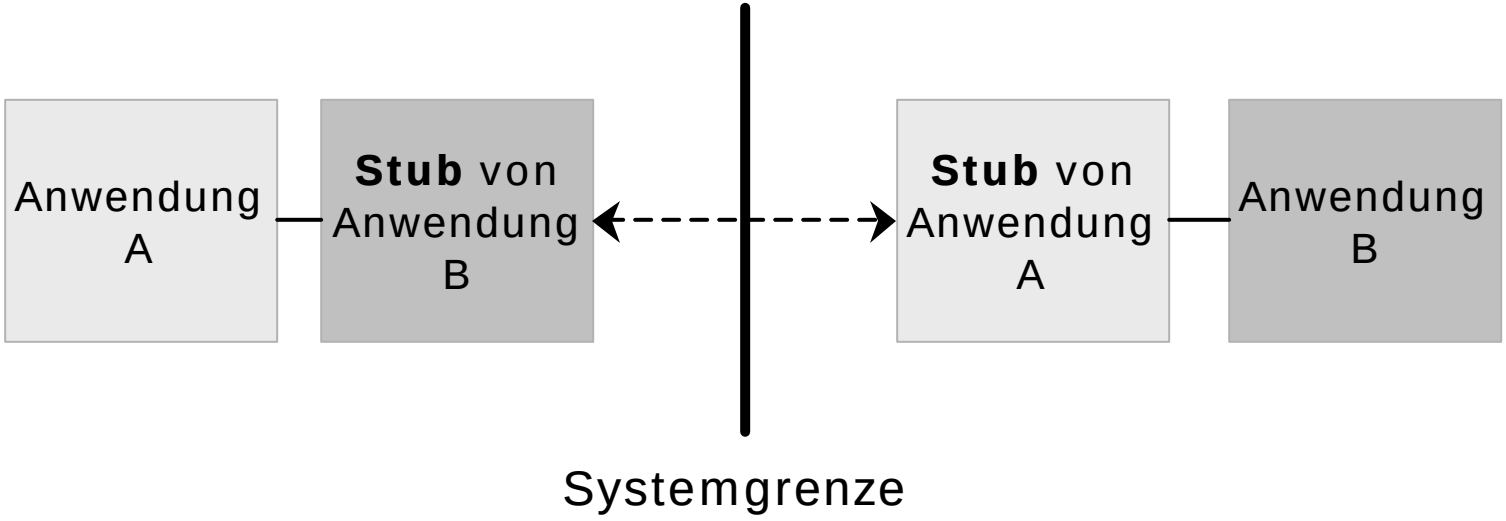
- IPC-Modelle: Dateien, Pipes, Sockets, Semaphore, shared memory
  - außer Sockets keins so weit standardisiert, dass es plattformübergreifend eingesetzt werden könnte.
  - außer shared memory skalierbar und internetfähig
  - operiert auf Bitebene -> zu kompliziert für komplexe Anwendungen

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, ...



Remote Procedure Calls

- 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 $10 \ldots 10^4$), 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.

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.

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

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)

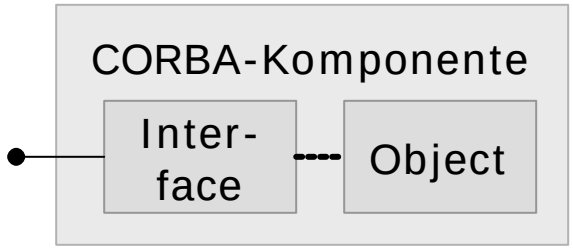
Schnittstellenspezifikation und Objekte

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

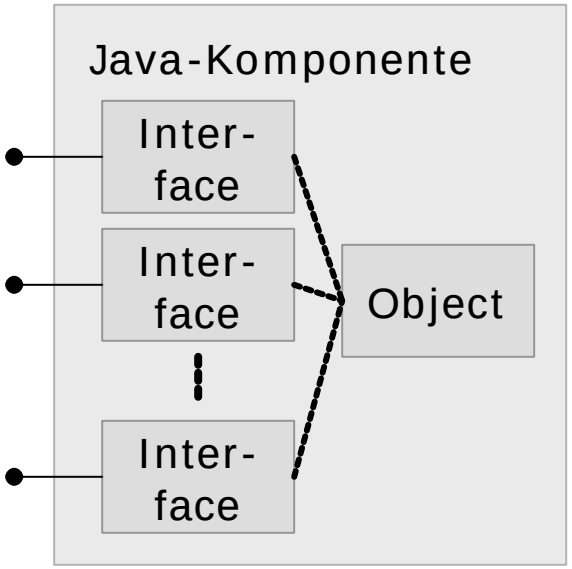
- **Schnittstellen-Beschreibung** durch IDL
  - mehrere Standards koexistieren (insb. OMG IDL und COM IDL)
  - Java und CLR: Keine IDL, sondern sprachspezifisches Meta-Datenformat, das auf jede der IDL abgebildet werden kann
    - dazu sind entsprechende Abbildungen zu spezifizieren
      - *Java to 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*.

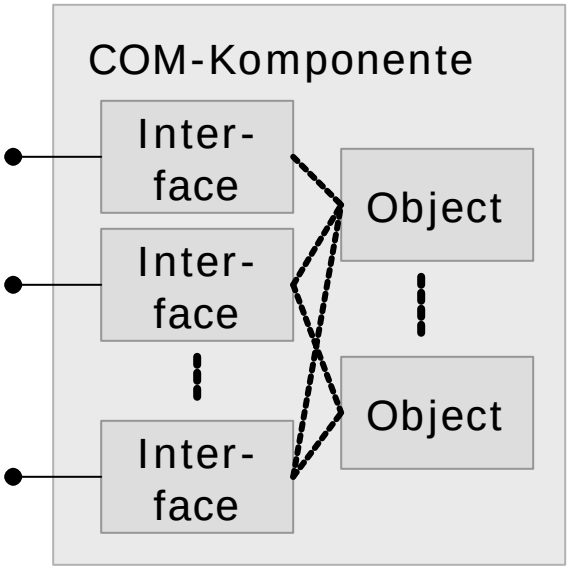
Bindung von Schnittstellen an Objekte

- Drei wesentlich verschiedene Ansätze:
  - 1 O ⇔ 1 S (CORBA 2, SOM)
    - Objekt speichert Programmzustand und Implementierung **seiner** Schnittstelle
    - Schnittstelle kann durch Mehrfachvererbung entstanden sein



- 1 O ⇔ * S (Java, CLR)
- * O ⇔ * S (COM)
  - Problem der Bindung von Schnittstellen an Objekte
  - Identität des Objekts muss geklärt werden





Schnittstellen und Vererbung

- mehrere Implementierungen derselben Schnittstelle möglich
- Implementierung kann mehr Funktionalität bereit stellen als durch die Schnittstelle definiert
- CORBA: Traditioneller Objektansatz
  - Jede 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

Schnittstellen und Vererbung

- Java: Objekte können mehrere Schnittstellen implementieren, aber stärker am Vererbungskonzept orientiert
  - Default-Implementierungen von Interfaces durch abstrakte Klassen
  - 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 sind unterschiedliche Namensräume

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

Erste Realisierung des Komponentenkonzepts
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

Die Ausdehnung auf das Web

- Idee: Einbettung von beliebigen Objekten in HTML-Seiten
  - z.B. Java Applets
- 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)

Die OMG und CORBA

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

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.

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 ORB's**
  - **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

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 3.0, Juli 2002
  - aktuelle Version CORBA/IIOP 3.0.3, März 2004 (http://www.omg.org)
  - bisher kaum Implementierungen, die CORBA 3 voll unterstützen