Vorlesung
Component Ware und Web-Services

- 3. Komponentenkonzepte -

Dr. Hans-Gert Gräbe, F. Schumacher
Wintersemester 2003/2004

Inhalt

Konzepte

1. **Zur Geschichte des Komponentenkonzepts**
2. **COM – Microsofts dokumentenzentrierter Ansatz**
3. **CORBA – der geschäftsprozesszentrierte Ansatz der OMG**
4. **Java – Suns webzentrierter Ansatz**

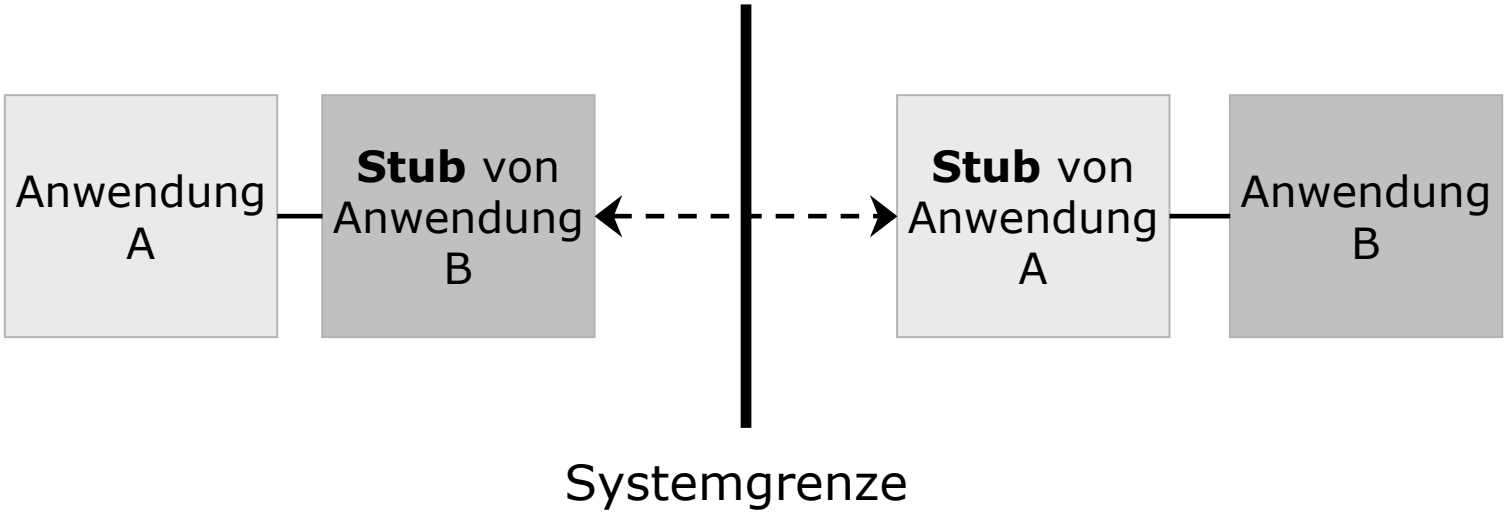
Zur Geschichte des Komponentenkonzepts

- **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.

Zur Geschichte des Komponentenkonzepts

- **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, ...



Zur Geschichte des Komponentenkonzepts

- **Remote Procedure Calls** (RPC, 1984)
  - Nutzung leichtgewichtiger RPC zur IPC auf derselben Maschine (Windows NT)
  - Vorteil: einheitliches Abstraktionsniveau für alle Kommunikationserfordernisse (innerhalb eines Prozesses, zwischen Prozessen, zwischen Computern)
  - Nachteil: Versteckt Kommunikationskosten
    - Unterschied um Faktor 10 ... 10^4

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

Zur Geschichte des Komponentenkonzepts

- **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 RPC zwischen heterogenen Plattformen konsistent einzusetzen.

Zur Geschichte des Komponentenkonzepts

- **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
  - Methoden müssen dazu genügend (binär kodierte) Information bieten, die durch Introspektion zur Laufzeit abgefragt werden kann
  - geschieht heute meist in speziellen Laufzeitumgebungen
    - Java Virtual Machine (Java)
    - System Object Model (IBM)

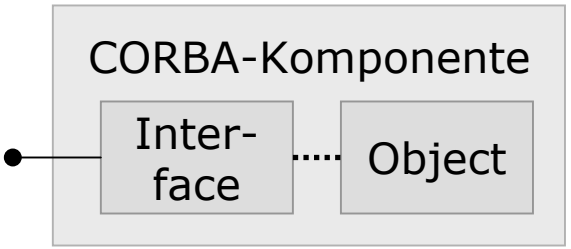
Der RPC-Ansatz ist deutlich aufzubohren, wenn mit Objekten und Methoden mit laufzeitabhängigem Verhalten umgegangen werden soll.

Zur Geschichte des Komponentenkonzepts

- **Über RPC-Mechanismus hinaus gehende Fragen, die von einem Komponentenkonzept beantwortet werden müssen:**
  1. Spezifikation von Schnittstellen (interface)
  2. Behandlung von Objektreferenzen, wenn der lokale Bereich verlassen wird
  3. Auffinden von Diensten
  4. Behandlung der Evolution von Komponenten (Versionsmanagement)

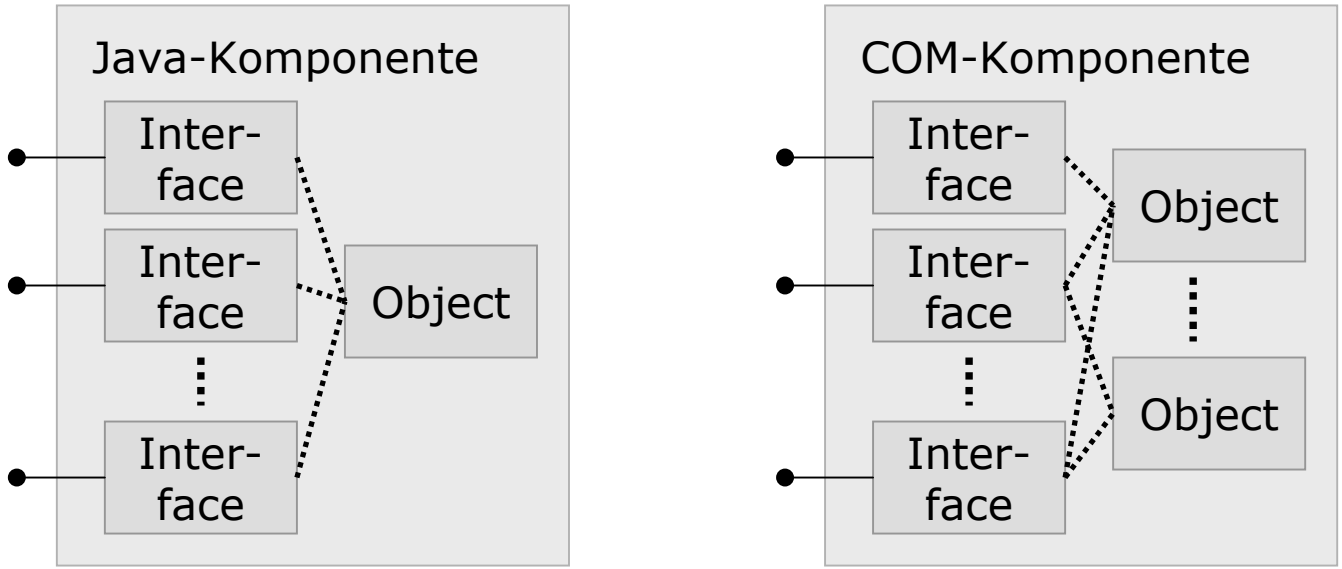
Zur Geschichte des Komponentenkonzepts

- **Schnittstellenspezifikation**
  - Interface = abstrakter Datentyp
    - Sammlung von Operationsbezeichnern mit ihren Signaturen
    - Signatur = Typ und Aufrufmodi der Parameter
  - Drei Modelle zur Bindung von Schnittstellen an Objekte innerhalb einer Komponente
    - 1 O <-> 1 S (CORBA)
      - aber: die eine Schnittstelle kann durch Mehrfachvererbung entstanden sein



Zur Geschichte des Komponentenkonzepts

- **Schnittstellenspezifikation**
  - Drei Modelle zur Bindung von Schnittstellen an Objekte innerhalb einer Komponente
    - 1 O <-> * S (Java)
    - * O <-> * S (COM)



Zur Geschichte des Komponentenkonzepts

- **Schnittstellenspezifikation**
  - Problem der **Schnittstellenidentifikation** innerhalb des Objekts
  - **Schnittstellen-Beschreibung** durch IDL
    - mehrere Standards koexistieren !
  - **Polymorphismus** wird unterschiedlich unterstützt
    - CORBA: Jede Komponente (=Objekt) hat nur ein Interface, das aber dynamisch erweitert werden kann
    - COM: Komponente hat mehrere Schnittstellen in einer Liste
      - Schnittstellen bedienen mehrere Objekte
      - Interface unveränderbar, aber Schnittstellenliste einer Komponente kann dynamisch erweitert werden
    - 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

Zur Geschichte des Komponentenkonzepts

- **Auffinden von Diensten**
  - Dienste werden über ihren Namen identifiziert
    - UUID als Standard der Open Software Foundation
    - COM (Microsoft) verwendet modifizierte Version: Global Unique IDentifier (GUID)
  - Über den Namen muss wenigstens folgende Funktionalität abrufbar sein:
    - Typtest der Schnittstellen zur Laufzeit
    - Introspektion der Schnittstellen zur Laufzeit
    - dynamisches Erzeugen neuer Objekte

Der dokumentenzentrierte Ansatz

- **Erste Realisierung: 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)
    - Visual Basic und VBX (Microsoft)
  - Weiterentwicklung durch Einbeziehung beliebiger Komponenten
    - OLE (Microsoft)
    - OpenDoc (Apple)
  - Webzentrierte Weiterentwicklung im Konzept der Plugins und Applets

Der dokumentenzentrierte Ansatz

- **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

Der dokumentenzentrierte Ansatz

- **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)