

Vorlesung Software aus Komponenten

2. Grundlagen

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2011/12

Komponenten und Management

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 wird es, 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 Komponenten-Kontrakt zu fixieren
 - Komponente als Einheit der Fehlerbehandlung

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

K. als Einheit der Kostenrechnung

- wichtig in größeren industriellen Kontexten, um Projektentwicklungskosten verfolgen zu können

K. als Einheit des Managements

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

Begriffsbildung: Im Gegensatz zum OO-Ansatz wird die Unterscheidung zwischen Programm und Daten wieder stärker betont

- Verhalten wird von **Komponente** bestimmt, gehört zur Designzeit und in die Verantwortung der Komponenten-Entwickler
- Attributwerte sind in **Objekten** gekapselt, werden zur Laufzeit geändert, Interpretation der Werte gehört zur Welt der Komponenten-Nutzer

Praktische Granularitäts-Ebenen von Komponenten:

- Ebene einzelner Desktop-Anwendungen – Intraprozesskommunikation, Zusammenbinden einzelner Funktionalitäten zu einem Gesamtprozess
- Ebene Middleware – Interprozesskommunikation, Zusammenspiel mehrerer Prozesse, auch auf verschiedenen Rechnern, zur Erfüllung einer Gesamtaufgabe / Anwendung (unser weiteres Thema)
- Ebene Geschäftsprozessmodellierung – Zusammenspiel mehrerer Anwendungen innerhalb eines betrieblichen Informationssystems (Thema der VL „Betriebliche Informationssysteme“ usw.)

Vorlesung Software aus Komponenten

3. Komponentenmodelle

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2011/12

Komponenten-Modelle

1. Erste Komponentenansätze
2. Grundlagen: Kommunikationskonzepte
3. OMG und CORBA, Common Object Request Broker Architecture
4. Sun und Java, Servlets und JSP, EJB
5. Microsofts und .NET, Common Language Runtime
6. Neuere Komponenten-Frameworks

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)

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 \Rightarrow Container für beliebige Anwendungen
- Kontrolleinheit \Rightarrow Dokumentenserver
- Container können hierarchisch ineinander geschachtelt werden

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

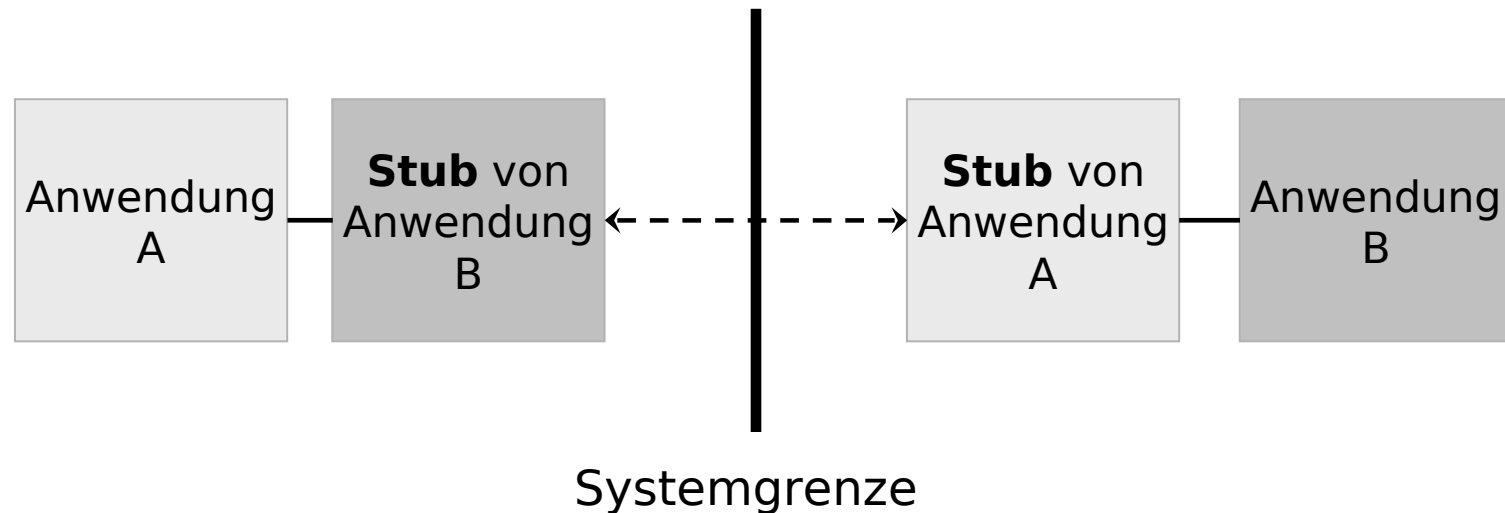
Interprozess-Kommunikation (IPC) auf OS-Ebene

- 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. Das ist zu kompliziert für komplexe Anwendungen

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

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, ...



3.2 Kommunikationskonzepte

Remote Procedure Calls

- Windows: Nutzung leichtgewichtiger RPC zur IPC auf derselben Maschine, unterstützt den Distributed Computing Environment (DCE) Standard der Open Software Foundation (OSF).
- 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).

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.

Weitere Fragen, die ein objektorientiertes Kommunikationskonzept jenseits eines so aufgebohrten RPC-Mechanismus beantworten muss

1. Wie werden Schnittstellen spezifiziert? Hier interessiert vor allem die programmiersprachen-unabhängige technisch-funktionale Spezifikation der Aufrufcharakteristika
2. Wie werden Dienste aufgefunden und bereitgestellt?
3. Wie wird die Evolution von Komponenten gehandhabt? (Versionsmanagement)

Zu 1.: Schnittstellenspezifikation

Definition: Interface ist ein abstrakter Datentyp

- Sammlung von Operationsbezeichnern mit ihren Signaturen
- Signatur = Typ und Aufrufmodi der Parameter

Schnittstellen-Beschreibung durch spezielle Sprache: 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, siehe <http://www.omg.org>
 - dazu sind entsprechende Abbildungen zu spezifizieren
Java to IDL language mapping specification (Version 1.4 vom Jan. 2008)
 - **Umgekehrt:** Java-Werkzeug **idlj** erzeugt aus einer (OMG) IDL-Beschreibung (u.a.) ein Java *interface* (Version 1.3 vom Jan. 2003, formale Spezifikation im pdf-Format Jan. 2008)

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

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 (Juni 2002),

- 1154 Seiten pdf-Dokument,
siehe <http://www.omg.org/spec/CORBA/index.htm>

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 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

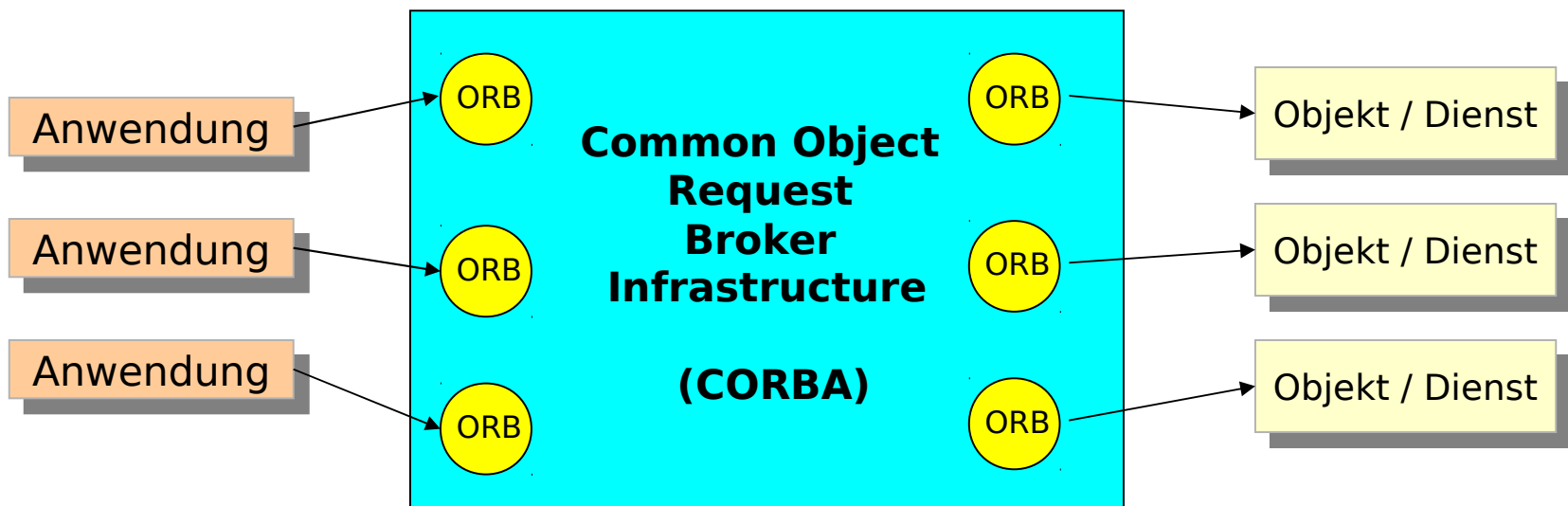
Etappen der Entwicklung von CORBA (Fortsetzung)

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 3.1.1, August 2011
(<http://www.omg.org/spec/CORBA/index.htm>)
- bisher wenig Implementierungen, die CORBA 3 voll unterstützen

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



Laufzeitbindung von Methodenaufrufen

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

