Vorlesung
Software aus Komponenten

2. Grundlagen

Prof. Dr. Hans-Gert Gräbe
Wintersemester 2013/14

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten und Systemdesign

Komponenten und Analyse

- Analyse komplexer Systeme erfolgt durch Zerlegung in handhabbare Einheiten, Tiefenanalyse der Einheiten und Zusammensetzung von Extrakten der Analyseergebnisse zum Gesamtsystem
- Ansatz: white box → black box
- Entwurfsexpertise lässt sich auf diese Weise kapseln und wiederverwenden (design expertise ready for use)
- Komponentenzuschnitt längs dieser Abstraktionsgrenzen erleichtert das Verständnis des Gesamtsystems

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten aus technischer Anbietersicht

Komponenten und Compilierbarkeit

- Compilierbarkeit kann aus Sicht der Auslieferung interessant sein
- Compilierbarkeit und Analyse sind eng verbunden
  - white box: Analyse
  - black box: Compilierbarkeit
- Compilierbarkeit und Optimierung
  - Optimierung ist ein globales Phänomen, deshalb Antwort 1: Einheiten möglichst groß wählen
- Antwort 2: Optimierung *zwischen* Komponenten, vielleicht sogar erst nach der Lokalisierung
  - Verfahren muss allerdings im Komponentenkonzept und in der Komponentenbeschreibung verankert sein

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten als Einheit der Packung

Komponenten und der Auslieferungsprozess

- 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

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

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

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 2012/13

UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

Komponenten-Modelle

1. Erste Komponentenansätze
2. RPC als Middleware Kommunikationskonzept
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

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)

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 und .NET (Microsoft)
- CORBA (Object Management Group)
- Java und Java-Frameworks (EJB, Spring, OSGi)
- Webservices als lose gekoppeltes Konzept

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. Das ist zu kompliziert für komplexe Anwendungen

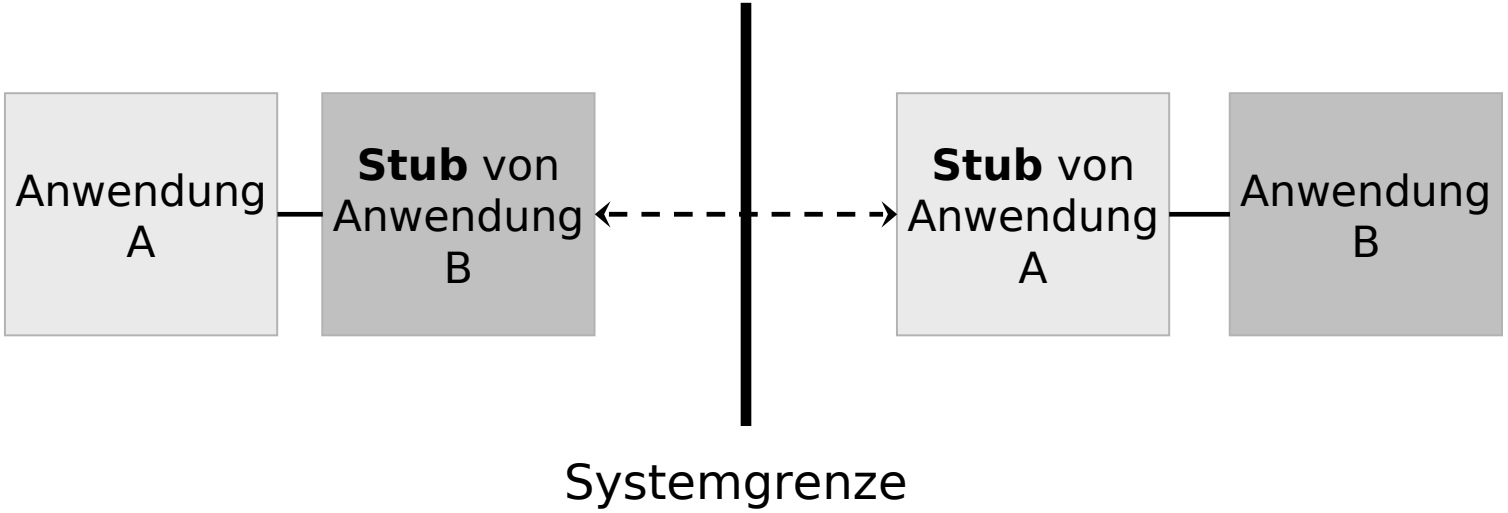
IPC operiert auf Bitebene und ist deutlich zu kompliziert für komplexe Anwendungen.

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, ...



UNIVERSITÄT LEIPZIG
Institut für Informatik
Betriebliche Informationssysteme

- 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

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

Weitere Fragen, die ein objektorientiertes Kommunikationskonzept 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)

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 (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**